Online Monitoring and Computational Steering of Massive Parallel CFD Simulations

Vom Fachbereich Informatik der Technischen Universität Kaiserslautern
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
genehmigte Dissertation
von

Christian Wagner

Tag der wissenschaftlichen Aussprache: 19. Juli 2013

Dekan:	Prof. Dr. Arnd Poetzsch-Heffter
Vorsitzender des Prüfungsausschusses:	Prof. Dr. Peter Liggesmeyer
1. Berichterstatter:	Prof. Dr. Hans Hagen
2. Berichterstatter:	Prof. Dr. Gerik Scheuermann

D (386)

Acknowledgements

I want to take the chance to thank all the people who have supported me during my research time and helped this dissertation coming into existence.

First of all, I would like to thank my advisors Andreas Gerndt, Charles Hansen, Christoph Garth, and Hans Hagen. In particular, I would like to thank Andreas for the opportunity to visit the German Aerospace Center for three and a half years and for his great support during this time. I am thankful for the support of Charles Hansen and Christoph Garth, who especially made my visits in Utah and California a fruitful time for my scientific research as well as for many experiences. I am especially thankful to Prof. Hans Hagen, who gave me the opportunity to join the International Research Training Group "Visualization of Large and Unstructured Data Sets - Applications in Geospatial Planning, Modeling, and Engineering" (IRTG 1131). Being member of his group allowed me to conduct my research in an international and interdisciplinary environment.

Furthermore, I would like to thank all my friends and colleagues for their intellectual exchange and personal support. First of all, I am very thankful to Markus Flatken for his great collaboration during my time in Braunschweig. Without our common work on hybrid rendering, major parts of this thesis would not have been possible. Furthermore, I want to thank my colleagues Rolf Westerteiger and Fang Chen for providing a friendly atmosphere that help me during this time.

Last, but most important, I would like to thank my family, my parents, and especially Kerstin for their confidence and endurance during this period. They always believed in me and my work and knew to cheer me up when times were hard.

Zusammenfassung

Aufgrund enormer technischer Fortschritte von hochperformanten Computersystemen und numerische Weiterentwicklungen ist computergestützte Simulation heute eines der wichtigsten Werkzeuge moderner Ingenieure. Mittlerweile werden experimentelle Methoden zur Untersuchung komplexer physikalischer Phänomene mehr und mehr durch Simulationen ersetzt. Während die Rechenpower moderner Computersysteme die Simulation immer komplexerer Systeme und Gleichungen ermöglicht, stellt die Verwaltung der ständig wachsenden Datengrößen durch höhere zeitliche und räumliche Auflösungen jedoch neue Herausforderungen dar. Um numerische Simulationen auch in Zukunft effizient durchführen zu können müssen die verursachten Hardware- und Energiekosten minimal gehalten werden. Jedoch werden komplexe Simulationen oft mehrfach ausgeführt, weil verschiedene Fehlverhalten zum Neustart der kompletten Simulation führen können. Computational Steering, die Interaktion mit laufenden Simulationen, versucht hier steuernd einzugreifen und Neustarts zu verhindern. Die Menge an produzierten Daten lässt aber verschiedene Lücken klaffen zwischen der Menge der Daten, die berechnet werden können und die Menge der Daten, die verarbeitet werden können. So ist zum Beispiel die reine Speicherung aller erzeugten Daten unmöglich geworden. Der Schwerpunkt dieser Dissertation liegt auf der Entwicklung neuer Methoden die die Steuerung, Exploration, Visualisierung und Analyse laufender numerischer Simulationen erlauben.

Im ersten Teil dieser Arbeit wird ein Software-Framework vorgestellt, welches die Steuerung existierender Simulationen vereinfachen soll. Es erlaubt Simulationen mit verschiedenen Visualisierungsalgorithmen, Rendering-Systemen und Interaktionsmethoden zu erweitern. Das vorgestellte System wird anhand einiger Beispiele demonstriert, darunter eine Methode um die Netzgüte hochauflösender Rechengitter während der Laufzeit zu analysieren um evtl. Netzadaptionen zu initiieren.

Der zweite Teil behandelt das interaktive Online-Monitoring großer Simulationen. Mit zunehmender Größe kann das reine Kopieren von Rohdaten zwischen Simulationen und Visualisierungsclustern sehr zeitaufwändig werden. Daher präsentiere ich hier eine in-situ Schnittflächen-Methode, die unnötige Datentransfers vermeidet,

jedoch weiterhin interaktive Update-Raten ermöglicht. Somit wird es ermöglicht, interaktive Schnittflächenvisualisierungen in einer virtuellen Umgebung auszuführen.

Werden komplexer Visualisierungen benötigt, wie zum Beispiel eine Skalarfeld-Visualisierung mittels Iso-Flächen, wird das Rendern mit interaktiven Bildwiederhohlraten problematisch. Im dritten Teil meiner Dissertation behandle ich ein hybrides Rendering-Verfahren, welches auch für große Daten eine interaktive Exploration in einer virtuellen Umgebung sicherstellen soll. Während Renderaktivitäten mit hoher Renderlast auf einem entfernten parallelen Renderer ausgeführt werden, wird die Visualisierung mit einer einfachen lokalen Kontextgeometrie gemischt. Dieses Kapitel diskutiert die Adaption der entfernt gerenderten Bilder auf die aktuelle lokale Ansicht, behandelt Image-Streaming für die Darstellung auf großen Displaywänden und verbesserte Interaktivität sowie paralleles Rendern von zeitabhängigen Daten.

Schließlich behandelt der vierte Teil die Feature-Extraktion in großen Strömungssimulationsdaten. Feature-Extraktion abstrahiert komplexe Feldeigenschaften und zeigt ihre signifikanten, strukturellen Komponenten auf. Während existierende Methoden typischerweise auf komplexen, mathematischen und topologischen Strukturen oder anwendungsspezifischen Definitionen basieren, werden in diesem Kapitel die Konzepte der harmonischen Analyse zur Detektion, Extraktion und Klassifizierung verwendet.

Abstract

Due to tremendous improvements of high-performance computing resources as well as numerical advances computational simulations became a common tool for modern engineers. Nowadays, simulation of complex physics is more and more substituting a large amount of physical experiments. While the vast compute power of large-scale high-performance systems enabled for simulating more complex numerical equations, handling the ever increasing amount of data with spatial and temporal resolution burdens new challenges to scientists. Huge hardware and energy costs desire for efficient utilization of high-performance systems. However, increasing complexity of simulations raises the risk of failing simulations resulting in a single simulation to be restarted multiple times. Computational Steering is a promising approach to interact with running simulations which could prevent simulation crashes. The large amount of data expands gaps in the amount of data that can be calculated and the amount of data that can be processed. Extreme-scale simulations produce more data that can even be stored. In this thesis, I propose several methods that enhance the process of steering, exploring, visualizing, and analyzing ongoing numerical simulations.

In the first part of this thesis, a software framework is introduced flexible steering capabilities to existing numerical simulations. This framework covers system-design aspects of computational steering solutions in order to couple simulation codes with visualization algorithms, rendering systems, and user interaction methods. Its useful application is demonstrated by some examples, including an approach to interactively analyze the quality of high-resolution meshes in order to steer mesh adaptation online.

The second part focuses on interactive online monitoring of large-scale simulations. With increasing data set sizes, copying raw simulation data between simulation and visualization clusters can be very expensive and time-consuming. An in-situ cut-plane approach is introduced which avoids unnecessary data transfers and still provides interactive update rates required to be useful in highly immersive virtual environments.

In visualization scenarios involving more complex features, such as iso-surfaces, rendering high-resolution simulation data at interactive frame rates becomes challeng-

ing. The third part of my dissertation introduces a hybrid rendering approach which enables the usage of virtual environments for interactive exploration. While heavy rendering workload is performed on a remote parallel rendering solution, the visualization is enriched with locally rendered context information at high update rates. In this chapter, I discuss the adaption of remote images to local images, image streaming for display walls with high screen resolutions, and introduce further improvements on interactivity and parallel rendering for time-dependent data sets.

Finally, the fourth part focuses on feature extraction methods to enable the visualization of large computational fluid dynamics data sets. Feature extraction abstracts highly complex fields by depicting significant structural components. While existing methods are typically based on mathematical complex topological structures or application specific feature definitions, I apply the concepts of harmonic analysis towards the extraction, detection and classification of features.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Traditionally, numerical simulation and analysis in computational fluid dynamics (CFD) is a sequential process. To prepare a simulation run, the flow domain is discretized by generating a simulation mesh. In order to perform a simulation on a cluster system, this mesh is divided into partitions, transferred to a cluster or supercomputer and the result is transferred back. Then, a variety of post-processing tasks, such as scalar- or vector-field visualization techniques, should give insight to the physical problem. In this traditional approach, parameters chosen wrongly cannot be identified until the post-processing step. Thus, in error cases the simulation has to be re-run with tweaked parameters. This is an iterative process that can be time consuming, especially if one iteration lasts more than a few days. Computational steering is aiming at reducing the simulation times by shortening the time used to identify wrong parameters results in high productivity enhancements. In computational steering many open problems need to be solved, e.g. how to monitor a running simulation effectively, what are sufficient interaction methods, and how should a software architecture look like to support a flexible steering process [MvWL98].

In a system-design aspect, computational steering solutions inherently combine coupling of simulation codes with visualization algorithms, rendering systems, and user

interaction methods. Therefore, a suitable software framework is needed. In the past, computational steering systems were developed to interact with ongoing simulations by enhancing existing visualization tools [EFG+05, PWJ97] or by developing specific computational steering frameworks [JB10]. These solutions mainly concentrate on data management and data interfaces [CDE03]. However, the main drawback of both approaches is that all steerable parameters as well as callable methods have to be known at compile time [WFM+10].

In the Computational Steering process, Online Monitoring is a key component. To be able to steer a running simulation, a user needs to explore its current state [MvWL98]. Existing online monitoring solutions mainly focus on adapting parallel rendering approaches and seldomly on providing interactive explorative solutions as well as dealing with different hardware setups [JB10]. According to [Tom06], visual exploration is the process of giving an overview of data and by allowing users to interactively browse through different regions. Here, the term interactivity is defined by two requirements, the rendered frame rate and the system response time. The frame rate is the update rate of the image presented to a user and should not be lower than 10 Hz [Bry96], or 30 Hz in virtual environments [KBLH03], to prevent flickering images. Both state that the system response time should not exceed 100 milliseconds, otherwise users cannot match input and systems response anymore. In order to detect characteristics of the data as quick as possible, this explorative analysis is running directly on raw simulation data [SM99]. However, in the case of monitoring large-scale numerical simulations data is distributed over different computer resources and different existing techniques have to be combined in order to meet the required interactivity conditions. Therefore, in-situ co-processing is used processing data already in the memory of the simulation compute nodes in order to shrink the data to be send. Even pre-processed data can be too large to be transferred and multi-resolution data formats can help to provide interactive renderings and stream visualization data to the visualization system. Even, if the visualization is preliminary and details are increased with time, the user remains interactive in using the visualization.

1.2 Overview and Contribution

My work covers a range of research topics associated with computational steering of computational fluid dynamics simulations and can be grouped into the following categories: (1) system design, (2) interactive online monitoring, (3) interactive hybrid rendering, and (4) feature extraction. I will cover each of these topics in one of the following chapters.

Chapter 2 addresses system-design aspects of steering computational fluid dynamics simulations. I introduce FSSteering [WFM+10], a flexible and domain-specific approach. Two examples demonstrate the usability of this framework. Furthermore, a more elaborate example is presented [CWF+12], supporting online mesh quality evaluations. My research contributions in this chapter are:

- Section 2.2
  - Description of FSSteering, a light-weight computational steering framework to enhance existing numerical simulations.
  - Successful steering of two example applications with FSSteering. Existing simulations can be improved by interactively adapt their mesh, if necessary, or can be enhanced by any user command to implement run-time steering of the simulation behavior.
- Section 2.3
  - Description of an approach to interactively analyze mesh quality online.
  - High-resolution mesh quality information is reduced in-situ into a presentation that can be handled interactively by the visualization frontend.
  - Successful interactive exploration in order to find critical simulation mesh regions.

Chapter 3 focus on interactive online monitoring of large-scale numerical simulations useful to inspect early simulation phases. My approach presented here [WGHH12] is aiming on large-scale simulation scenarios in which copying raw simulation data

between simulation and visualization clusters can be very expensive. Therefore, I present an in-situ cut-plane approach which avoids any unnecessary data transfer.

The exploration of scalar fields with the help of cut-planes requires the interactive movement of the plane along the data set. However, in the examined distributed environment, the required update rates can not be achieved with classical cell-based extraction methods.

- This chapters contributions are:
  - Implementation of in-situ cut-plane extractions based on point sampling. I present three different variations providing a progressively refined visualization. Thus, the result scales from preview to high quality images depending on parameter values and available computing power.
  - The presented cut-plane extraction methods provide built-in time thresholds. Therefore, they are capable to fulfill weak real-time requirements of highly interactive systems like virtual reality environments.
  - The implementation into the FSSteering framework enables to reuse simulation data. Therefore, only small amounts of extra memory is required which allows the application to a wide range of simulations.

Chapter 4 deals with interactive rendering aspects of high-resolution simulation data. I introduce a hybrid rendering approach [WFC+12] that enables the interactive exploration of large-scale simulation results in an interactive fashion in virtual environments. While intermediate simulation results can easily exceed the capabilities of virtual environment hardware, heavy rendering workload is outsourced to a remote parallel rendering solution. This chapter discusses this approach in detail as well as extension to large-display walls and time-dependent simulation data. My scientific contributions are:

- Section 4.2
  - Implementation of a hybrid rendering technique to overcome insufficient local rendering capabilities of virtual environment. The performance and

scalability of parallel rendering solutions is exploited to enhance a local rendered context geometry with highly detailed visualizations.

- Reducing the drawbacks of remote rendering solutions, namely high latency and low frame rates, by adjusting available remote images to current local renderings.
- Introduction of an image-based adjustment method based on adaptive point-sizes, which successfully fills in surface holes and does not fill in suspicious background pixels.
- Successful implementation on a three-pipe powerwall system.
- Detailed performance and image quality benchmarks of the presented approach.

- Section 4.3

  - Implementation of a hybrid rendering technique in order to overcome insufficient local rendering capabilities of display wall visualization clusters. The techniques allows for interactive navigation through a large-scale dataset or online monitored simulation data.
  - Multi-resolution, based on z-order curves, supports quick overviews.
  - Introduction of a progressive image streaming approach which adds only low latency to the remote rendering process.
  - Using progressives streaming instead of usual sub-sampling requires no additional render passes for different resolutions and adapts automatically to network capabilities.
  - Successful presentation on a display wall with twelf display tiles.
  - Detailed performance benchmarks of the presented approach.

- Section 4.4

  - Presentation of a modified hybrid rendering pipeline that enables for handling large time-dependent data sets.

  - Interactive animation of local geometry which enables for grasping complex motion patterns.
  - Combination with complex features supporting valuable context information.
  - Introducing additional post-processing steps to the remote rendering process without loosing interactivity.
  - Enabling for rendering multiple time-steps in parallel resulting in enhanced remote update rates.
  - Demonstration with a simulation data set of the SHEFEX I re-entry flight experiment. My approach enables to animate the flight body's motion at interactive frame rates in combination with shock wave rendering. Therefore, iso-values can be selected interactively and iso-surfaces are extracted and rendered on the fly.
  - Detailed performance analysis of the modified pipeline.

In chapter 5, I focus on feature extraction methods to enable the visualization of large computational fluid dynamics data sets. Especially for highly complex fields arising from modern computer simulations, visualization efforts can be reduced or made feasible by depicting significant structural components and their interactions, allowing for an abstracted view.

Existing methods are typically based on topological structures or application specific feature definitions. While the former leverage a deep mathematical framework to generate a topological skeleton of a field and are uniformly applicable to general fields, the latter requires intimate knowledge of the application domain. Here, I apply concepts of harmonic analysis towards the extraction, detection and classification of features [WGH12].

- This chapter contributes with:
  - Illustration of the typical global approach to harmonic analysis and identification of the problems for global approaches.

- Definition of a local approach more feasible computationally in contrast to global approaches.
- Definition of a low dimensional feature vector based on a feature space over small neighborhoods.
- Definition of a local feature strength measurement, based on properties described in Laplacian eigenbases.
- Numerical comparison of two direct possibilities for the choice of the required Laplacian's discretization.
- Discussion of numerical aspects of the computation of large eigenvalue sets.

Chapter 6 concludes my dissertation by summarizing the presented work and discussing possible topics for future work.

Chapter 2

Manipulation for Computational Steering and Online Monitoring

Numerical simulation is a common tool for modern engineers used in order to gain insights into complex flow situations. Therefore, computational fluid dynamics simulations are set up which requires the specification of specific parameters. After these simulations have been performed by a cluster or supercomputer, post-processing algorithms extract physical features and generate visualizations for visual feedback. Many parameters chosen wrong can only be identified at that point, and eventually the simulation has to be done again with modified parameters. With iteration times of days or even weeks, methods with higher productivity are desirable for providing quick research insights. Therefore, the ability to check whether a simulation is still on track is important. If possible, critical simulations should be guided towards a reasonable state by applying changes during run-time.

To tackle this situation, computational steering systems were developed to interact with ongoing simulation runs. Most of the available computational steering environments are enhanced visualization tools and concentrate on providing meaningful visualizations. A complicated instrumentation of simulation codes is needed to make data available to those systems. In opposition, native computational steering frameworks concentrate on easy simulation coupling with minimalist interfaces. However, they often support limited visualization and analysis techniques which have to be

implemented by the user. In general, main drawback of both approaches is that all steerable parameters as well as callable methods have to be known at compile time.

As a part of this thesis, I developed a domain-specific approach called *FSSteering* to tackle the kind of problems mentioned above. All details of how to enhance existing CFD simulations by computational steering capabilities are described in section 2.2. *FSSteering* has been developed as an extension to the German Aerospace Center's computational fluid dynamics system called FlowSimulator. Based on my results simulation scripts can be made steerable by users. The easy-to-use interface requires nearly no instrumentation. Furthermore, simulation data is available to post-processing algorithms without data conversion. My *FSSteering* extension interprets steering commands to be executed and only reports unknown commands to the simulation. This enables the execution of tasks or changing parameters provided by the FlowSimulator environment without either being known to the simulation script or having to be implemented by CFD engineers. For example, one can change or adapt the underlying mesh of any CFD simulation during run-time which results in better simulation convergence without changes to the simulation setup or script.

In section 2.3, application examples are presented utilizing my *FSSteering* framework. These examples clearly demonstrate the success of my results. The first presented example is focusing on the simulation grid as a key element of numerical accuracy. In the second example, the angle of an airplane model is changed during run-time as an example of model specific steering actions.

Section 2.4 deals with online monitoring aspects of large-scale numerical simulations. There, I present an interactive mesh exploration tool which enables the inspection of the quality of high resolution meshes. Further online monitoring aspects and details are discussed in depth in chapter 3.

2.1 State-of-the-Art and Related Work

Since steering simulations have been met with interest for many years now, a lot of work has been done. An overview of earlier systems can be found in [MvWL98]. Online monitoring is essential to identify in what way a simulation has to be steered.

Therefore, most steering systems concentrate on visualization or are enhanced visualization tools, like [PWJ97] and [EFG+05]. Native frameworks like [JB10] are available to enable for computational steering, but have a high adaptation overhead to specific problems. [CDE03] uses XML descriptions of simulation scripts to handle data and concurrency at instrumentation points. Only few existing systems try to tackle domain-specific requirements; one CFD-specific adapting the simulation grid resolution is presented in [KTH+02].

The FlowSimulator [ME10] is an open and efficient framework to unify massively parallel and multidisciplinary CFD simulations, providing a unified interface for incorporated tools. This is achieved by a layered approach. The FlowSimulator DataManager (FSDM) forms the common backbone and provides a common interface to store and exchange data in memory. Written in C++ it provides a number of classes providing functions typical for CFD-related numerical simulations. Using the automatic interface generator SWIG [Bea96] all of FSDM's interfaces are also provided in Python.

Explorative and interactive visualization is supported using the VRFlowVis application, a visualization frontend for steady and unsteady CFD data sets based on ViSTA and ViSTA FlowLib [SGvR+03]. ViSTA allows the frontend to scale from simple desktop systems to high-end immersive VR environments. ViSTA FlowLib is a specialized library that provides particular interaction methods [WBK07] [WHS+06] and efficient rendering techniques for working with time-dependent CFD data.

Simulation features are extracted from the raw data and mapped to visualization components by a post-processing application based on Viracocha [GHW+04] [WSK+07]. It is decoupled from the visualization front end and distributed to High Performance Computing (HPC) resources. Visualization features are extracted in parallel, and as soon as first results are available, the extracted geometry data is sent back to VRFlowVis to be rendered.

2.2 FSSteering – A Computational Steering Architecture

My computational steering architecture presented in this chapter aims on enabling already existing computational fluid dynamics simulations to be steered. They should be steerable without much impact on the existing simulation scripts. Therefore, I developed *FSSteering* as an extension to the FlowSimulator system. By this, my framework is able to provide easy access to existing functionality, and simulation scripts can be coupled with parallel post-processing back-ends as well as front-end systems.

In my target workflow different computing systems should be connected on-the-fly, cf. figure 2.1. A supercomputer or cluster system is performing a set of simulation tasks in batch processing. To steer one of the running simulations on-demand, different front end and back end systems need to be attached on-the-fly. Therefore, my framework supports a flexible connection topology dealing with heterogeneous networks, which is a clear advantage over previously existing solutions.

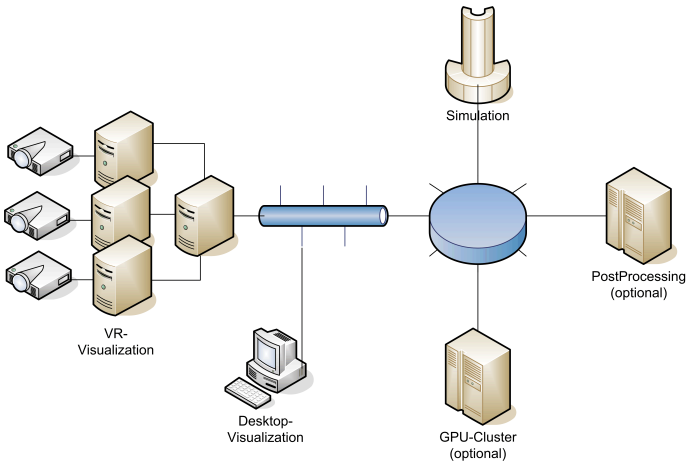


Figure 2.1: CFD simulations can be connected by multiple post-processing and front-end visualization systems on-demand. TCP/IP- as well as MPI-connections can be used for data communication.

2.2.1 System Architecture

The overall architecture of my *FSSteering*-framework can be seen in figure 2.2. Although I make use of the scripting interface offered by FlowSimulator, performance-critical tasks need to be implemented efficiently. For this reason, I implemented a core module in C++ which provides connection handling and data transfer methods. Access to these functions is provided by lightweight APIs, likewise available in Python and C++. In addition, the Python-API is bound to the FlowSimulator-API which allows to inherit its functionality and provide it to the connected applications via command requesting. In the run-time examples in section 2.3, I use both APIs, Python and C++.

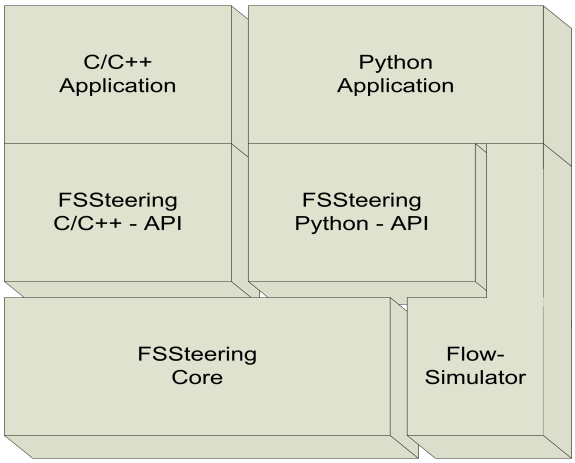


Figure 2.2: *FSSteering*'s main functionality is implemented in the core module and made accessible by lightweight APIs. The Python-API can also use functions offered by FlowSimulator.

2.2.2 Run-Time Execution

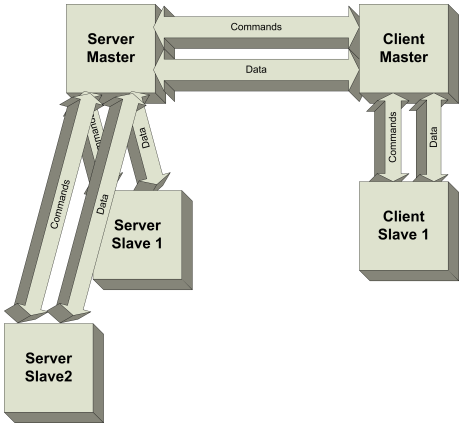
At run-time, my *FSSteering* framework makes every steerable simulation act as a server waiting for the client to connect. Clients act as servers for further connections as well. This allows my framework to support arbitrary topology between different distributed systems. The examples I present in chapter 4 clearly demonstrate this

advantage. There, a remote rendering system can be connected on-demand in order to support interactive exploration in a virtual environment.

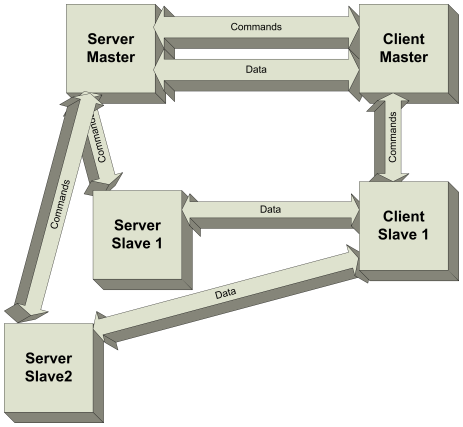
A connected client sends commands to the simulation server and waits for response. A set of predefined system commands exists for registration and updating variables and sending geometry or field data over connections. Calling domain-dependent FlowSimulator functionality, such as mesh adaptation, offers the possibility to change simulation behavior without being implemented in the application scripts in the first place. All commands unknown to *FSSteering* are assumed to be user commands and are returned to the caller, e.g. the simulation script. In order to ensure simple handling, commands are represented as Python dictionaries including necessary parameters, and are mapped to dictionaries of strings in the C++-API. Commands are sent through the system in a serialized representation. Their interpretation occurs when triggered by the simulation.

The execution of commands is based on message queues. For command execution centralized request management [EDC04] is used, a simple, yet efficient synchronization scheme. All commands are gathered at a client's master node and are sent to the server's master node. When a simulation triggers the processing of upcoming commands, the server broadcasts all new commands to the server's slaves. This choice perfectly fits to the SPMD (single program, multiple data) programming model used in FlowSimulator scripts.

Special care was taken in managing different connections in order to provide a flexible connection topology. Although command communication is always gathered and distributed through the master nodes, this does not hold for data communication. As depicted in figure 2.3, in addition to 1-to-1 connections via master-to-master connection, it is possible to establish n-to-m connections, where each server node is connected to an arbitrary client node. This setting is used in the steering examples of section 2.3. For general purposes, geometry and field data can be sent as raw binary data, the VTK file format is also supported.



(a) 1-to-1-connection: Data and commands are gathered at master nodes and redistributed to slave nodes.



(b) n:m-connection: While commands are gathered and redistributed, data is distributed in parallel.

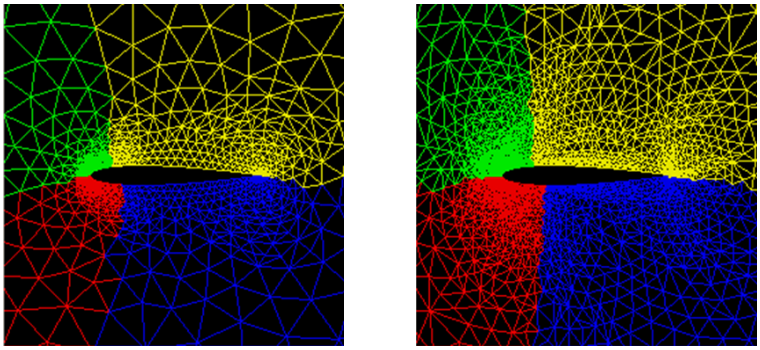
Figure 2.3: Commands are always gathered and redistributed in master nodes. For data transmission, each data node can be connected to arbitrary client nodes.

2.3 Computational Steering Results

This section demonstrates the effective usage of my *FSSteering* framework. In the examples, I made a FlowSimulator simulation steerable using the *FSSteering*-Pyhton-API, running on four computational nodes. The parallel post-processor Viracocha connects to the computational nodes via a n-to-m connection, one processing node to each simulation node. Simulation and post-processor are controlled using a ViSTA front-end. In this setup two frequent steering applications are demonstrated.

2.3.1 Numerical Steering

Since the underlying simulation mesh is essential for numerical convergence to physical meaningful results, the additional possibility to influence the mesh during runtime can prevent restarting simulation runs. Figure 2.4 shows the effect of additional adaptation runs initiated by my *FSSteering* framework in the FlowSimulator environment. Note that no additional code adjustment was required since mesh adaptation is one of the algorithms provided in FlowSimulator and *FSSteering*.



(a) Initial mesh. (b) Adapted mesh.

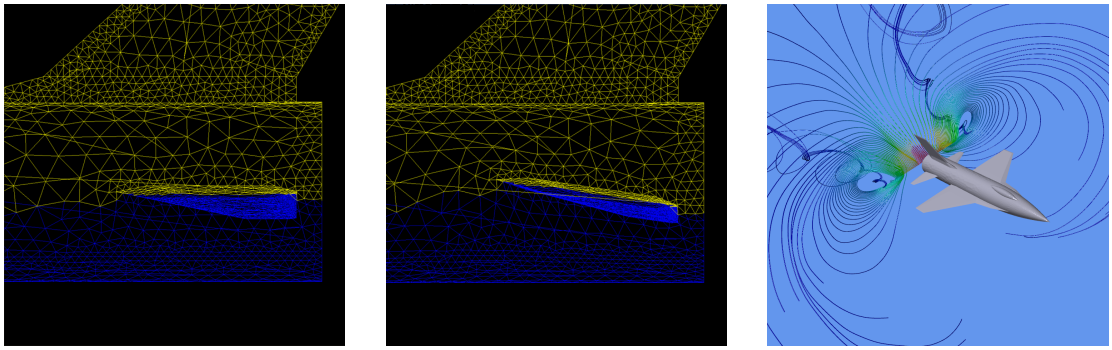
Figure 2.4: For an initial computational mesh (a), background adaptation is triggered improving numerical accuracy (b).

2.3.2 Simulation Steering

Contrary to the first example, this example shows how a simulation script is enriched with a user-defined code, see figure 2.5. The used simulation script has the ability to change aileron, rudder, and elevator angles in a synthetic aircraft model. *FSSteering*'s abilities to schedule user-defined steering commands during run-time is used to successfully deform the mesh. Mesh deformation is controlled and viewed by the front-end application. Two wire-frame visualizations and a virtual reality view of the front end is shown in figure 2.5.

2.4 Interactive Online Mesh Exploration

Current Computational Fluid Dynamics simulations such as the design of airplanes are dealing with complex scenes that consist of large-scaled meshes. With the increasing computation power supported by new hardware, these simulations are also growing towards extreme and even exa scale. Due to their complexity and overwhelming sizes, these simulations can last for weeks even on large-scale cluster computers. However, simulation processes often crash or fail to converge due to poorly defined meshes, resulting in a complete restart of the whole simulation.



(a) Initial configuration. (b) Changed angle. (c) Explorative view with changed angle in VR front-end.

Figure 2.5: In this simulation steering example a command to change the elevator angle was triggered, (a) and (b). The influence on the flow field is analyzed in the explorative visualization environment, (c).

Mesh quality is one of the key factors in the success and accuracy of computational fluid dynamics simulations. Since current simulations are paving the way towards extreme scale computations, mesh quality exploration in an interactive manner becomes challenging. Due to the fact that large-scale simulations are extremely time-consuming and likely to fail within a certain time frame, an immediate on-the-fly mesh analyzing tool is strongly desired. Such tools should not only provide a visualization of mesh quality at an interactive frame rate, but also allow the user to detect and locate regions of the meshes that lead to computational error, in order to apply further mesh refinement while the simulation is still running.

In this context, I present an interactive mesh exploration tool in a virtual environment. My combination of interactive exploration and mesh quality evaluation in a user-defined region of interest results in an enhanced analysis tool. The example presented in this section clearly demonstrated the advantages of my approach. Online monitoring of running numerical simulations is discussed in more detail in section 3, where I examine in-situ cut-plane extractions.

2.4.1 Interactive Mesh Quality Analysis

Mesh visualization in a virtual environment enables a detailed and immersive exploration of the mesh quality. The immersion of the display system enhances the perception of depth, thus allowing a rapid localization of certain mesh regions.

Providing interactive frame rates is critical in virtual environments. Not only the amount of the data to be visualized poses a challenge to interactive frame rates, but also the requirements of providing on-the-fly visualization to an ongoing simulation.

The potential and usage of immersive virtual reality for scientific visualization have been studied in a general way by [vDFL+00, vDLS02]. Applications of combining these two research areas are for example VIRPI [GSRB01], and a point-based mesh visualizer presented by [GSJ+06]. However, little literature has addressed the problem of visualizing extreme-scale data sets in VR. [Ma07, ARS11] have pointed out the upcoming problems and implications in visualizing ever increasing data set sizes.

My interactive mesh exploration tool tackles the issues mentioned above. To address the problem of data size and obtain an interactive frame, I propose a processing pipeline in order analyze a mesh error function, facilitating extreme scale visualization techniques with user-defined regions of interest, as well as error threshold functionality. As a result, simulation experts are able to monitor the simulation process on the fly and apply mesh refinement without restarting the simulation or redefine the meshes.

2.4.2 System Design

In this section, my framework and mesh analysis approach is presented, which is based on my *FSSteering* framework.

Figure 2.6 shows the architecture. A large-scaled CFD simulation is running parallel on a cluster of simulation machines. The major goal of my approach is to provide the user a possibility to analyze the mesh quality at the current simulation time. This will allow the user to identify regions of bad quality determined by an error-threshold value. This value can then be sent to the simulation, thus allowing for mesh refinements, and carry on the simulation process further on the modified mesh. The

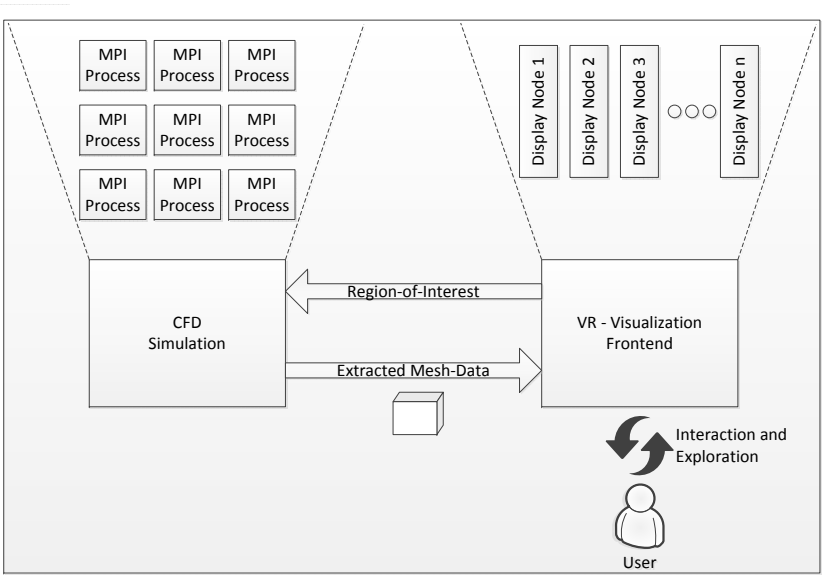


Figure 2.6: Architecture: A VR visualization front-end is connected to the ongoing CFD simulation running on the back end simulation cluster. The user can interactively define the region of interest. This region of the mesh will be extracted from simulation and sent to the front-end to be rendered.

major challenge for my proposed framework is to provide an interactive visualization of the mesh quality, allowing users to quickly track down regions of bad mesh.

The benefit of doing immersive visualization for extreme scaled meshes is merely to provide details of the large data. The size of the simulation mesh makes it impossible to perceive and explore mesh features on a conventional desktop display. Not only detail and occlusion are an issue, a 2D desktop display will also not meet the need to interactively explore the mesh data. As [vDFL+00] point out, people can more readily explore and understand complex structures by peering around them or handling them. By applying mesh quality visualization in a virtual environment, the user will be able to explore the mesh data in a more natural way and allocate errors in an interactive and rapid manner.

2.4.3 Simulation Mesh Quality

In CFD simulations, mesh quality is a key factor in determining the convergence and success of the solver. Locating poorly defined meshes at an early stage can save the simulation experts weeks of time by preventing a simulation to crash. Due to the nature of numerical solvers, computations are performed on a group of neighboring cells. Therefore, if one mesh cell is poorly defined, not only the mesh cell itself but also its neighboring cells need to be refined.

A large body of research on defining mesh qualities and metrics, such as [Knu01, Ber99], can be found. Many mesh quality metrics are per-element-based, such as [Dur99]. It allows the user to identify a single mesh element which has a low quality.

The approach I present here makes use of a per-edge measurement of mesh quality:

$$Error_{edge} = (v(P_1) - v(P_0))||P_1 - P_0||^2$$

with $v(P_i)$ being the pressure coefficient of the field at point p_i. Per cell, the error is taken as the maximum of the edge errors, which is

$$Error_{cell} = \max\{Error_{edge}\}.$$

This is a common metric for scalar valued fields. However, it can be substituted by any other metric more suitable, depending on the current application scenario. The visualization techniques I present in the following are independent from that.

2.4.4 Visualization Techniques

This section elaborates on my choices of visualization techniques as well as the reasons for using them.

Mesh quality metrics allow the user to identify a single mesh element which has a low quality. However, this type of approach will not be applicable in case of extreme-scale mesh data sets. The overwhelming number of the mesh cells poses a severe occlusion problem and a single mesh cell can not be distinguished anymore.

Furthermore, cell volumes can vary by some orders of magnitude within a single data set and the size of critical cells can be much smaller than the display pixel sizes. Therefore, critical information might be lost during rendering.

To avoid the front-end from being overloaded with data and minimize the amount of rendering time for the visualization, I facilitated the following visualization paradigms in order to provide a meaningful and intuitive representation of the mesh quality. Figure 2.7 shows my resulting prototype.

2.4.4.1 In-Situ Mapping of Error Values

Previous visualization of mesh qualities focused on visualizing a single element. However, when the mesh size is huge, one will not be able to pick out a single mesh cell from the entire data, not even with the help of large displays. Moreover, it is desired from the simulation experts that entire regions including poorly defined mesh should be modified. Another important fact while visualizing extreme-scale data sets is that the size of a single mesh cell might be even smaller than a pixel size on the screen.

To address these issues, I re-mapped cell quality metrics onto a uniform-grid. In order not to miss any important information, each voxel of the uniform grid contains the maximum error value of all underlying cells. In my presented *FSSteering* framework, this mapping is done in-situ. Each processing node maps its local cells assigned by domain decomposition. The final uniform-grid representation is gathered on the simulation side. This presentation is very small compared to the original data set size and prevents the front-end from processing any raw data.

2.4.4.2 Region of Interest

To further limit the amount of mesh data to be visualized, I incorporate the region of interest method [PNP05] into our approach. Instead of re-mapping mesh quality information to a uniform grid spanning the whole spatial domain, a box of interest is implemented, with which the user can extract quality information in smaller areas.

The knowledge and experience of the user will speed up the allocation of poorly defined regions by constraining the search around a more-likely area.

2.4.4.3 Interactive Error Thresholding and Brushing

Finally, the exploration of mesh errors is performed in the virtual reality front end with standard volume visualization techniques [DCH88]. Initially, the full range of present error values is mapped to the range of rendered colors. However, the user is only interested in the range of high error values, defined by the selection of a certain error threshold. Brushing, as a common information visualization techniques, allows the user to interactively select a subset of the data in order to identify outlayers [RW06]. In this case, a brushing function is implemented allowing the user to define and limit the error range. As a result, only a minimal amount of the volume is mapped to opaque colors. Irrelevant information is filtered out and the region of the poorly defined mesh can be identified easily.

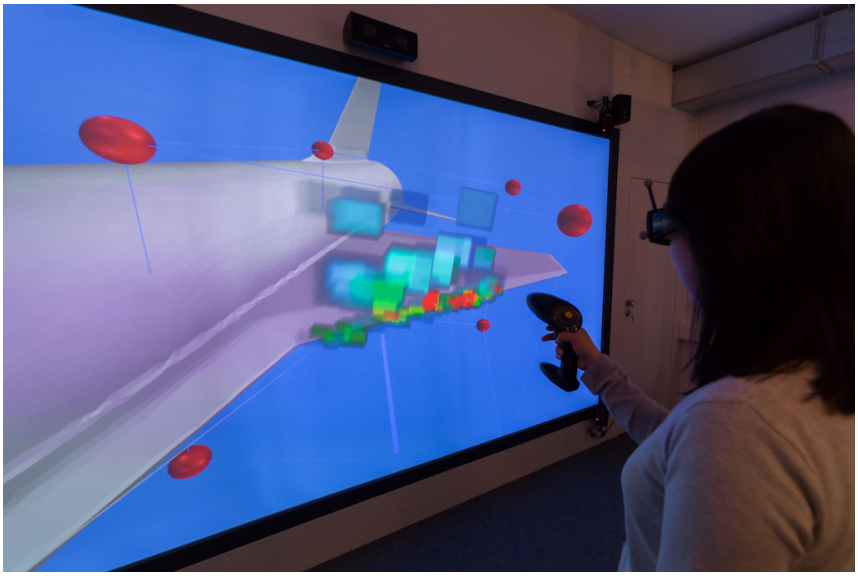


Figure 2.7: User interacts with the mesh data set in an virtual environment.

2.4.5 Results

To illustrate the usefulness of my approach, a test data set simulating the airflow around an airplane wing is used. The data set consists of $310,156$ tetrahedrons and approximately $60,000$ points. The mesh is partitioned into 4 blocks. The simulation is running on 4 compute nodes utilizing a workstation with an Intel Xeon E5520 @2.27GHz. Our in-situ approach is performed on the same workstation in order to extract the quality of the mesh. The workstation is connected via MPI to the virtual reality system on which the volume rendering is performed on a NVIDIA Quadro FX 5800 graphics cards.

One region of interest selected by a user covered approximately $50,000$ cells, thus $1/10$ of the original mesh. This region is then mapped into a $256x256x256$ voxel volume. Therefore, the size of a voxel exceeds the size of a mesh element, which further explains why one is interested in mesh regions rather than single elements. The total computation time required to extract the mesh quality inside the region of interest took about 0.75 seconds.

Figure 2.7 shows a user exploring the mesh quality of the airplane dataset in front of a large display wall. With my tool the user is able to smoothly navigate through the whole mesh and quickly identify and locate critical regions inside the mesh, here color-coded in red, which require improved mesh quality.

I demonstrate the advantages of error brushing in figure 2.8. In the left image, mesh errors in the range of $[0.05, 0.32]$ are mapped to different opaque colors. In fact, only the regions colored in red are of users' interest which contain cells with high error metric values. However, these regions are hidden behind and are hard to perceive. By limiting the opaque colors to a smaller range (Fig. 2.8, right), insignificant information is removed, leaving only the mesh region unveiled which is of bad quality.

This method can be used to interactively chose an error threshold which defines critical cells inside a running simulation. By triggering an adaptation inside the simulation, the mesh quality can be improved for further numerical iterations.

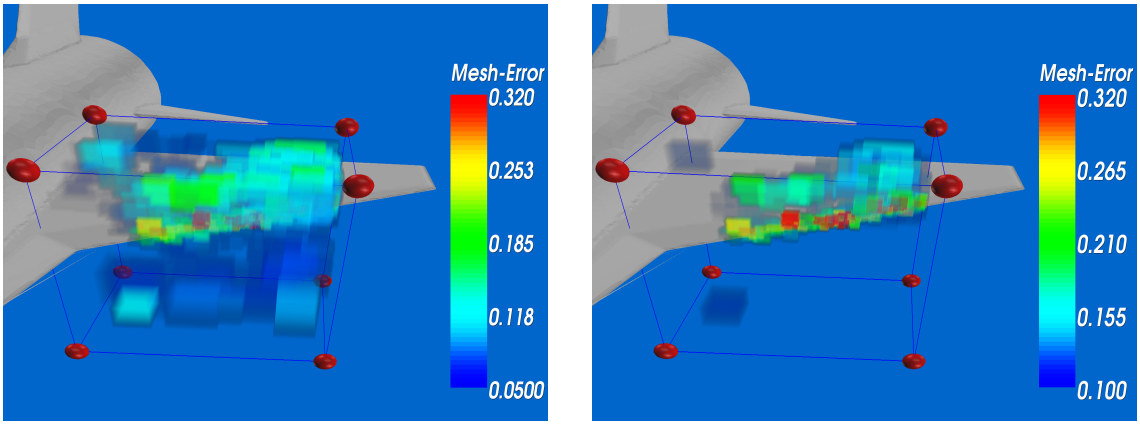


Figure 2.8: Left: volume rendering of the error function with chosen mesh cells. Right: User can iteractively select the error range to be mapped to volume. Error brushing allows the user to identier error outlayers and located regions of poor-defined meshes in a fast way.

2.4.6 Discussion

This section presented my interactive mesh exploration tool which utilizes virtual environments in order to monitor large-scaled simulations. The application provides a potential solution and framework to perform extreme-scaled mesh exploration. It was clearly shown that virtual environments can serve as a powerful tool to include a human into the loop of iterative simulation and visualization.

For future work, I plan to apply this approach to more complex data sets at a higher scale. Using my presented approach would improve the online monitoring and interaction even of extreme-scale simulations. The in-situ processing part of the presented approach is scaling with the domain decomposition of the simulation itself. However, varying sizes of the region of interest as well as the size of the uniform sampling grid might introduce new demands to the front end.

2.5 Conclusions

In this chapter, I focused on the manipulation for computational steering and online monitoring of numerical simulations. When running large-scale simulations, their

successful execution depends on many factors. Furthermore, long simulation times of weeks or months prevents simulation setups on an extensive trial-and-error basis. Here, computational steering methods are more promising, avoiding repetitive restarts of simulations. However, major challenges are: (1) providing a software architecture capable to monitor and steer a simulation, that is easy to use and has low overhead, (2) adequate visualization of simulation quantities and qualities, especially in unexpected simulation states, (3) the size of data sets, which requires sophisticated data handling in order to provide interactive exploration possibilities.

This section presented my *FSSteering* framework, a flexible computational steering environment to tackle those challenges. As an extension to the FlowSimulator framework domain-specific needs of CFD engineers are addressed, targeting on simple usage. I implemented a flexible connection and data management between simulations on the one hand and front end as well as post-processing back end modules on the other hand and demonstrated its usage. Therefore, I presented a collection of different application examples. In two steering examples I showed the steering capabilities, namely the simulation mesh and its adaptation, as well as interactive steering of model parameters, such as wing angles of an airplane simulation. Furthermore, I clearly demonstrated the advantages of my presented interactive computational steering framework for large-scale numerical simulations with a more enhanced online monitoring example. My tool enabled to explore for spatial regions in a computational fluid dynamics simulation in which mesh quality needs improvements. In order to reduce rendering requirements for the front end, high-resolution mesh quality information was re-sampled to a uniform grid at the simulation side. With this, my approach enabled for interactive exploration to identify critical regions.

Chapter 3

Interactive Online Monitoring in Virtual Environments

In large high performance computing simulations, it is often the case that the state of running simulation needs to be interpreted or validated. This task is known as online monitoring and can be used, for instance, to inspect preliminary simulation results in early simulation phases in order to test e.g. correct simulation setups. Effective online monitoring methods are also fundamental for computational steering, which incorporates both the state inspection of an ongoing simulation and the modification of simulation parameters during run-time.

Typically, large-scale numerical simulations are carried out on distributed memory cluster systems. A common way to inspect these simulations is to copy the current simulation data to a separate parallel visualization cluster which performs classical post-processing tasks. However, copying simulation data to a separate post-processor can be very time consuming. Therefore, in-situ processing is a promising alternative. This approach does not move raw data from the compute nodes to storage or to other nodes via network connections. Instead, key algorithms are executed on the same compute nodes, ideally on the same data structures, and only processed results are moved.

In this chapter, I focus on the advantages and challenges of using virtual reality techniques in order to support in-situ online monitoring tasks for large-scale numerical

simulations. In order to inspect simulation data I focus on the interactive exploration of scalar data utilizing a cutting plane approach. Cutting planes are one of the established analysis tools in many application disciplines and engineers learned how to apply them to their simulations. However, cutting planes only support information in a local region. In order to understand a complex physical phenomena, cutting planes are required to be placed at multiple positions inside the spatial simulation domain. Therefore, interactive movements of a cutting plane along the data set domain, as visible in figure 3.1, is an efficient way to explore the data set and to identify the regions of interest.

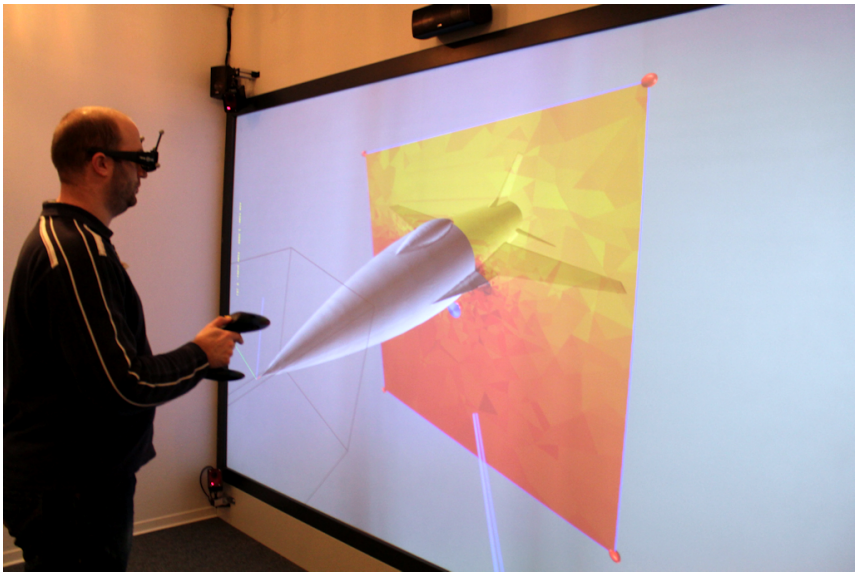


Figure 3.1: Interactive exploration of a running parallel CFD simulation with cutting planes using a virtual environment.

High update rates are required to achieve an interactive movement of the cutting plane. These update rates are easy to accomplish if the data set fits into the memory of a local graphics card and if the simulation grid is rectilinear. However, in my in-situ setup, cutting plane calculations have to take place at the simulation nodes, where raw data is partitioned and distributed, and extracted data needs to be transferred to the front end.

In most simulations, cell sizes vary over the computational domain and, therefore, the number of intersected cells change with the current cutting plane position. As-

sociated with the number of intersected cells extraction time as well as result size are influenced.

In this section, I will clearly demonstrate the benefits of virtual reality techniques for online monitoring of running large-scale numerical simulations. My approach uses cutting planes in order to explore and understand scalar fields of complex simulations. Even though cutting planes are useful in classical desktop environments, the intuitive interaction methods provided in virtual environments enhance their usefulness. An intuitive interaction with cutting planes allows exploring the simulation domain and makes it easy to find interesting regions. In order to remain interactive, high update rates of the information on the cutting plane are required. Since cell-based cutting plane extraction has limited extraction rates, I use a progressive sampling scheme. This sampling scheme guarantees certain update rates on the cost of image quality. In addition, I present two more local point sampling methods being able to evaluate even more sampling positions. Such methods allow interactive cutting plane extraction and update rates making them suitable to be used in highly interactive visualizations such as virtual reality environments. This support of interactivity was not provided in in-situ processing and online monitoring methods before.

3.1 State-of-the-Art and Related Work

Understanding the science behind large-scale simulations requires the extraction of meaning from datasets of hundreds of terabytes and more [MRH+07]. However, the cost of moving the simulation output to a visualization machine is increasing with larger simulations. According to [MWYT07], it is preferable to not move the data at all, or to keep the moved data to a minimum. This can be achieved by applying simulation and visualization calculations on the same parallel supercomputer *in-situ*, so that data can be shared.

3.1.1 In-Situ Approaches

A classification of in-situ implementations is presented in [RCMS]. In general, three different techniques can be used to couple a visualization or data-processing to a running simulation. These are:

Tight coupling: Simulation and data processing run on the same compute nodes. Therefore, they have direct access to the same data and no data movement is required at all. However, severe performance issues can occur. Data processing is required to scale at least as well as the simulation and sharing the same resources might lead to memory bottlenecks. Furthermore, simultaneous post-processing might slow down the simulation, because simulation computations have to wait for the post-processing to be finished. The solutions provided by ParaView [FMT+11] and Visit [EFG+05] are tight coupling approaches.

Loose coupling: In contrast to tight coupling, simulation and data processing do not share any resources. Raw data has to be copied from the simulation host to the processing resources over network. This transfer can be triggered either by simulation (push) or by data processing (pull). The advantage of this implementation class is that scalability and memory requirements of simulation and processing hosts do not have to match. However, data transfer can be a limiting factor. The ParaView plugin ICARUS [BSO+12] is coupling simulations loosely by memory mapping the HDF5 file format.

Hybrid coupling: Hybrid implementations combine tight and loose coupling approaches. Here, data is reduced in a tightly coupled processor and sent to a concurrent post-processor for further processing. The *FSSteering* framework, presented in this dissertation, allows for this type of coupling, as presented in chapter 4.

3.1.2 In-Situ Processing

In-situ data processing can be manifold. [MWYT07] categorizes the following processing steps that can be performed in-situ.

Data Reduction: Common data reduction techniques are sub-sampling, quantization and transform-based compression. Sub-sampling is the simplest way to reduce

simulation data. A common practice is to skip time steps and select, e.g. every hundredth time step. This creates a major challenge to temporal-space visualization and animation and further extraction methods like tracing of path-lines are becoming more inaccurate.

Simulation data is mostly computed in single or double precision floating point numbers with high accuracy. However, it is not always necessary to preserve this level of accuracy, for example, if relative values are in the focus of research and absolute values are not important. Also, hardware accelerated rendering makes use of texturing hardware with 8 or 16 bit of resolution. In those cases, quantization makes sense in order to reduce the amount of data to store.

Data quantization can be performed in many ways. Simple quantization methods are direct scalar quantization methods, which use only local data and are fast to compute. Data quantization also contains more elaborate methods making use of data statistics, such as the global Lloyd-Max method [GW06] or the local Jayant quantizer [Jay73].

Another class of quantization methods is vector quantization, that groups data values into blocks of data and encodes these blocks. Since these methods, such as the Linde-Buzo-Gray algorithm [LBG80], requires the training of a codebook, vector quantization methods are often too computationally expensive to be used as in-situ processing methods [FMA05].

Finally, transform-based compression is a very effective way to reduce data to store on disk. This compression transforms the data from spatial domain to frequency domain resulting in energy coefficients for each frequency. Since this representation is often more meaningful for the physical situation, a compression in this domain introduces less errors by only quantizing the less important lower energy coefficients more coarsely. Most popular transform-based encodings are the discrete cosine transform and the wavelet transform, later allowing an additional multiresolution data representation and a level of detail to be selected according to the visualization requirements. In terms of cost and performance, transform-based compression is a better choice for in-situ data reduction [MWYT07].

Feature Extraction: A feature is a particular physical structure isolated with domain knowledge. Some examples are vortices, shocks, eddies, critical points, etc. These features can be used to categorize the overall physical phenomenon. The saving in storage space with feature extraction can be very significant, however, scientists do not always know exactly what to extract and track in their data. [MM07] demonstrates a method for feature tracking using a low cost and incremental prediction and morphing approach to track a turbulent vortex flow. Feature extraction and tracking remains to be an active area of research, because the high-level data reduction explicitly takes into account domain knowledge. Although many feature extraction and tracking methods have evolved in the last decades, less work was done to apply them to in-situ processing.

Quality Assessment: Most of the presented in-situ processing methods focus on reducing data size during simulation run-time. Therefore, the information loss compared to the original data should be conveyed to the user to identify and quantify the loss of data quality. Most data quality metrics, such as the mean square error, require access to the original data and are therefore not applicable to large-scale simulations where the original data are too large. A solution applicable in in-situ processing is shown by [WWS+06], who only used statistical information extracted from the original data in the simulation. In the visualization the distance of the reduced data can be compared with the extracted statistical information and in order to indicate quality loss. An improved version [WM08] extracts statistical information in the wavelet domain also enables a cross-comparison of different reduction types.

Rendering: For monitoring and steering purposes a direct rendering of images in-situ can be beneficial to give insight into the simulation without requiring an additional visualization system. In [TYRg+06] in-situ rendering is conducted during a tera-scale earthquake simulation. For the presented ray casting visualization each processor renders its local data. The same data partitioning created by the simulation can be reused, and thus no data movement is needed among processors. Only an API provided by the simulation is required, because all access operations are read-only. No further changes are needed to adapt the simulation. In the image

compositing stage, a new algorithm is designed to build a communication schedule in parallel on the fly.

3.2 Distributed Cutting Plane Extraction

In order to perform interactive exploration of running simulations with cutting planes a crucial step is the calculation of cutting plane information in the first place. For interactivity purposes certain characteristics are required by the in-situ extraction algorithms. These are in particular:

- bounded runtime: To provide interaction, a minimal update rate is required. Therefore, the time between changing plane positions and rendering extracted results need to be bounded.
- preview images: In order to explore a dataset for each cutting plane position a representative image is required. If an exact result can not be determined in the requested time, a preview image need to be provided instead.

In the rest of this chapter, I will present a benchmark setup that I will use to analyze cell-based analytic methods and, furthermore, demonstrate the clear advantages of my presented approach.

3.2.1 Benchmark Setup

I use a real-world CFD simulation to evaluate my method. A typical airflow simulation around an airplane is used consisting of 13.6 million points and 35.2 million cells (12.7 million tetrahedrons, 22.5 million prisms). Cell sizes in this model are strongly decreased near the airplane wing. In a pre-processing step, points and cells are decomposed and distributed into 64 domains. The domain partitions are indicated by surface colors in figure 3.2.

The simulation is running on 8 nodes of a cluster system, each node equipped with an Intel Xeon 2.53 GHz quad-core processor and 48 GB RAM. The cluster's interconnect is a DDR InfiniBand network. The available virtual reality environment is

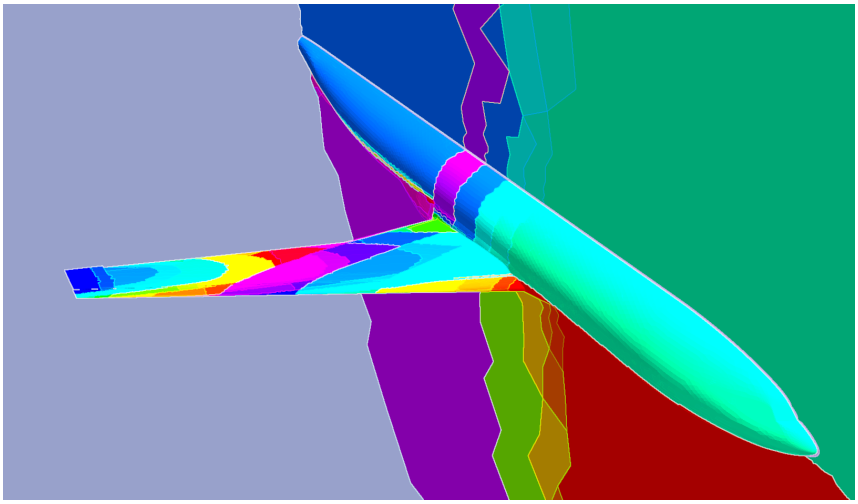


Figure 3.2: Airplane model used for benchmarking with 13.6M points and 35.2M cells. The model is decomposed into 64 domains for distributed simulation.

based on a three-pipe powerwall system and a Flystick interaction device. The visualization cluster driving the powerwall projectors consists of four nodes, each with a dual Intel Xeon 2.8 GHz quad-core processor, 24 GB of RAM and NVIDIA Quadro FX 5800 graphics card. The powerwall system is connected to the simulation via 1 GBit/s ethernet.

In order to ensure reproducible benchmark results, I defined a synthetic user scenario. In this scenario, depicted in figure 3.3, a cutting plane is moved along the airplane from its front tip to a position shortly behind the wings. During this movement a cutting plane needs to be extracted on 500 positions covering regions with low resolution as well as highly refined cells along the wing.

3.2.2 Cell-based Cutting Plane Extraction

First, I examine the behavior of cell-based cutting plane extraction algorithms, such as used by VTK and ParaView. Unstructured data sets are described as a set of basic cells, e.g. tetrahedrons and prisms, with values assigned to each vertex. The process of extracting cutting plane information is to determine all cells intersected by the plane and calculate the geometry information of the cut region. While the search for a set of possible candidate cells can be done very efficiently with spatial

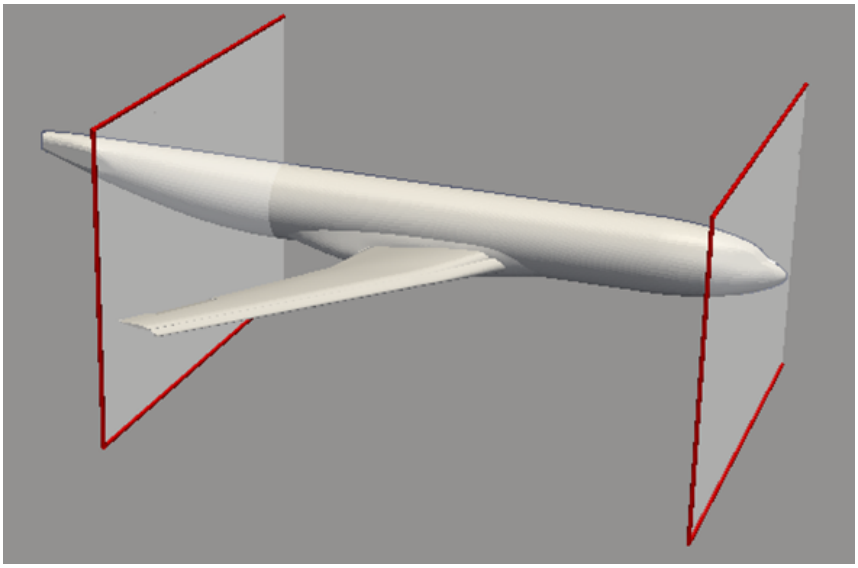


Figure 3.3: Benchmark scenario: cutting planes are extracted at 500 positions moving between the airplane tip and a position behind the wings.

search trees like octrees or binary space partition trees, the intersection has to be calculated for all candidate cells. The result is often represented as a triangulation.

This process is more or less the same in a distributed calculation, in which cells are partitioned in a pre-processing step and are distributed to a set of processing nodes. Cell searching and intersection can be done independently and in parallel on each involved processing node. In addition, the resulting two-dimensional cells are gathered to determine a complete cutting plane representation. The total run time is depending on the number of cells cut by the plane, since all intersected cells need processing. For unstructured grids the spatial distribution as well as the spatial resolution of cells can be a huge problem for efficient cutting plane extraction.

Since the number of extracted triangles can differ much for different cutting plane positions, extraction run time as well as output size are unpredictable, which is a problem for interactive rendering. This problem is demonstrated in figure 3.4, which shows the maximum run time of all compute nodes and the according number of intersected cells for each cutting plane position. Most important, the extraction time when intersecting highly refined regions can increase dramatically. In this benchmark, the maximum measured extraction time exceeded 20 seconds. These

high extraction times do not allow for the interactive exploration I am achieving in this chapter.

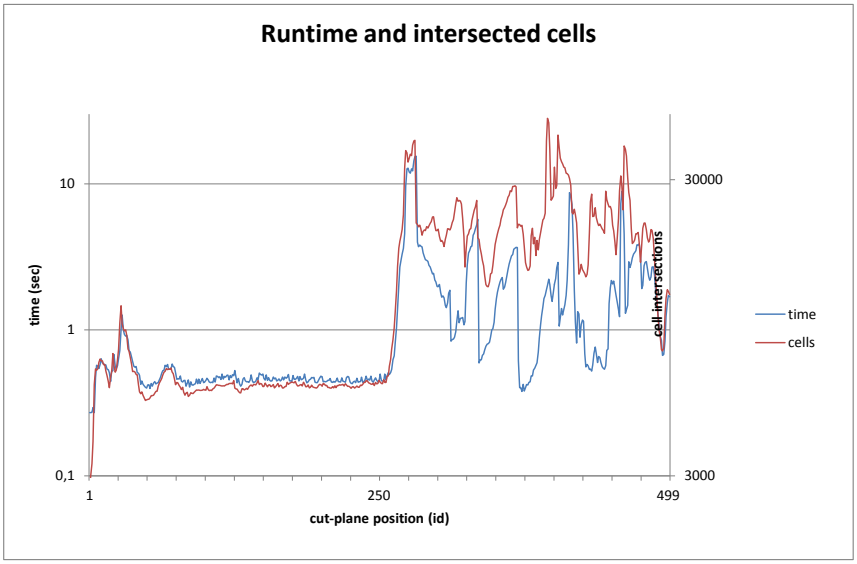


Figure 3.4: Maximum extraction time and corresponding number of intersected cells for cell-based cutting. This benchmark, performed by ParaView, shows a visible relationship between extraction time and number of intersected cells. The absolute time required at certain cutting plane positions do not allow interactive exploration.

3.2.3 Point-based Cutting Plane Extraction

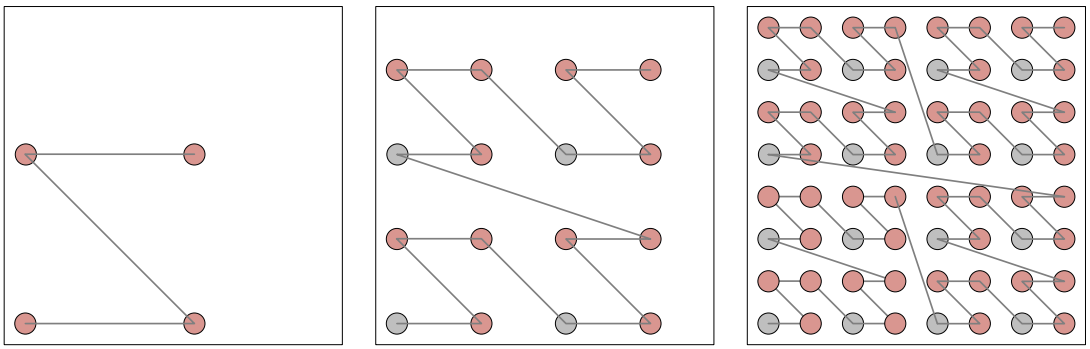
I clearly demonstrated in the last section, that cell-based cutting plane extraction algorithms can lead to very long as well as position-dependent extraction times. Therefore, the requirements for interactivity that I stated earlier can not be achieved by cell-based cutting plane extraction algorithms. In order to provide methods which fulfill those requirements, namely bounded run time and generation of preview visualizations, I will present point-based methods suitable for in-situ online monitoring.

I implemented my methods into my FSSteering framework, which was presented in chapter 2. This allows my approach to directly access data of ongoing simulations. Therefore, my method is able to extract cutting plane information in-situ without the need to copy any raw data.

3.2.3.1 Progressive Sampling

The last section demonstrated the strong correlation between cutting plane position and run-time of cell-based cutting plane extraction algorithms. To overcome this limitation, I use a progressive sampling strategy based on z-order space filling curves, here.

A z-order space filling curve, see figure 3.14, is a space filling recursive curve in which the first level is defined as a single point. By recursively substituting each vertex with a z-pattern, the z-curve is refined from one level to the next finer level. Therefore, each iteration adds three-times as many vertices as the prior iteration and after the i^{th} iteration the total number of vertices is 4^i.



(a) first level z-curve (b) second level z-curve (c) third level z-curve

Figure 3.5: A z-curve is a recursive pattern, in which finer levels are defined by replacing each point by four points in z-ordered shape: (a) first level, (b) second level, (c) third level.

I use this pattern to generate sampling points on the cutting plane in a regular pattern. Therefore, each compute node extracts point information for points covering their own partition domain and the results for each recursive level are gathered on the simulation master node. Figure 3.6 shows the result of sampling the domain at levels with different resolutions. Even in a low resolution, regions of interest around the wing can be identified.

Figure 3.7 shows timing results for my progressive sampling approach. Herein, the accumulated time is presented until point information for each progressive level extracted. This method has two important improvements over cell-based extraction

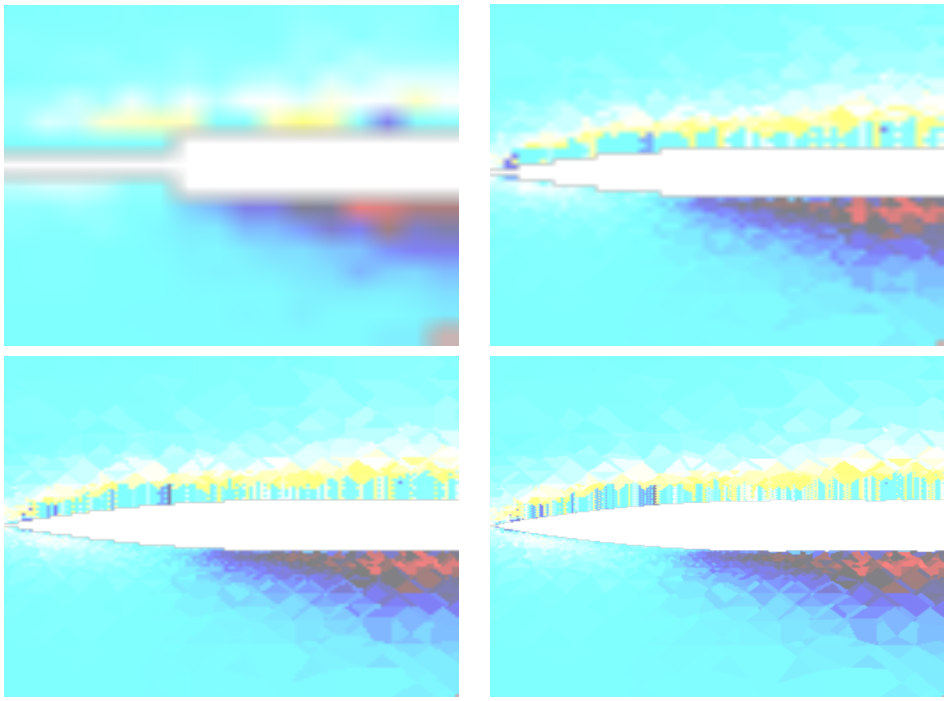


Figure 3.6: Four textures generated with progressive sampling (16x16, 64x64, 128x128 and 256x256 samples). Lower resolutions are sufficient to identify regions of interest.

algorithms. First, the impact of the cutting plane position and the number of the intersected cells is negligible. In contrast, extraction times in the cell-based approach benchmark differed by a factor of one hundred. And second, I achieved a representation possible to stream to the front end in a progressive fashion. Therefore, it is possible to stop streaming when a certain time threshold has elapsed. This results in a preliminary image possible to present to a user. Therefore, the extraction can be stopped at any time in order to request a cutting plane visualization for a new plane position, which is important for preserving interactivity.

3.2.3.2 Random Sampling

The disadvantage of progressive textures is their discrete resolution. One has to wait for all sampling points of a level to be calculated, before they can be streamed to the visualization front end. Furthermore, Each process needs a different time to find

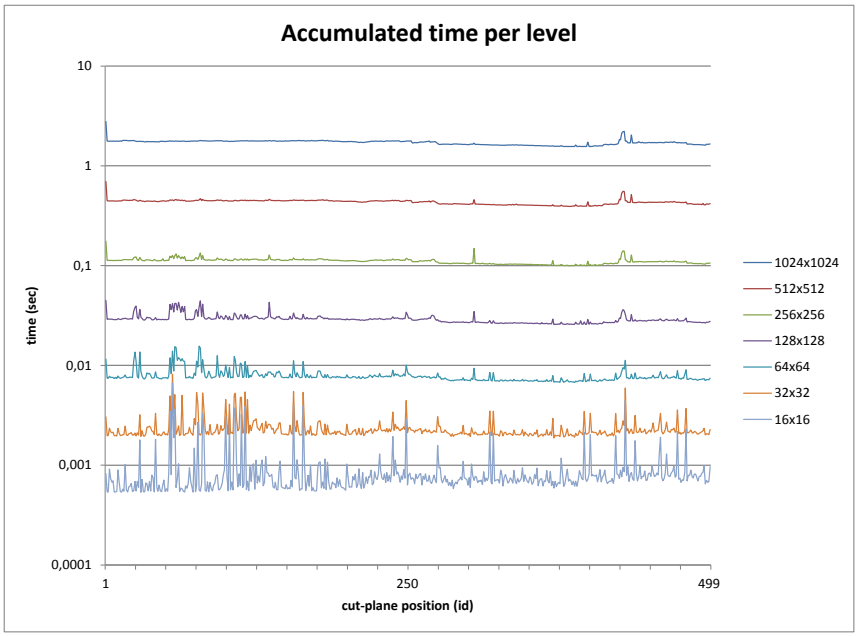


Figure 3.7: Total time to extract progressive texture levels. Even with increasing runtime in refined regions, strong variations as present in cell-based extraction are not noticeable.

and evaluate its part of a level. This results in unbalanced computations, because the whole progressive texture extraction is bounded by the slowest compute node.

To overcome this issue, my following two methods remove dependencies between distinct samples by applying pure point sampling. Here, sample locations as well as their sampled values are transmitted to the front-end. Since the calculation can be stopped after every point sampling, simple time thresholds can be used. To extract point information on each of the computing nodes, I apply following algorithm:

1. Determine the possible parameter space by restricting u,v-cutting plane space to bounding box of cells located on each processor. If parameter space is empty, stop.

2. While time threshold for computation is not passed and less than n_{bound} points have been extracted, sample at random position inside parameter space and extract the point values.

3. Gather extracted points at master node.

4. Select subset of points and transmit them to front-end.

The upper bound n_{bound} prevents the master node from processing too many points. It is a combination of the relative area according to the processing node and the maximal number of points transferable with the actual bandwidth to the front-end visualization system. It is defined as

$$n_{bound} = f * area_{proc}/area_{total} * points_{max},$$

where the factor f is a conservative factor, normally chosen between 1 and 2. During gathering, the actual subset of points to transfer needs to be selected. We do this by selecting $points_{max}$ points with a distribution according to the relative area.

To calculate point values, I use the CellTree [GJ10], a bounded interval hierarchy, to locate the cell containing the point. This search tree is optimized for uniform sampling and enables for fast cell location.

In figure 3.8, four images are shown demonstrating random point sampling. The bandwidth parameter $points_{max}$ is chosen with the values 1000, 2000, 3000 and 5000.

Figure 3.9 shows timing results for my random sampling approach. Herein, the number of sampling points per time is presented. This method improves over the progressive sampling approach. For example, after 100 ms, random point sampling is able to extract about 200,000 points, while the progressive level extracted in the same time has only 65,536 sampling points.

3.2.3.3 Adaptive Sampling

While random point sampling covers the cutting plane regions uniformly, no attention is given to the scalar field values. Regions with high variance in the scalar field are normally more important than regions with low variance. Therefore, I introduce an adaptive point sampling method with a probability distribution according to the variance of the scalar field along the cutting plane. We achieve this by adding variance information to the search tree used for cell location.

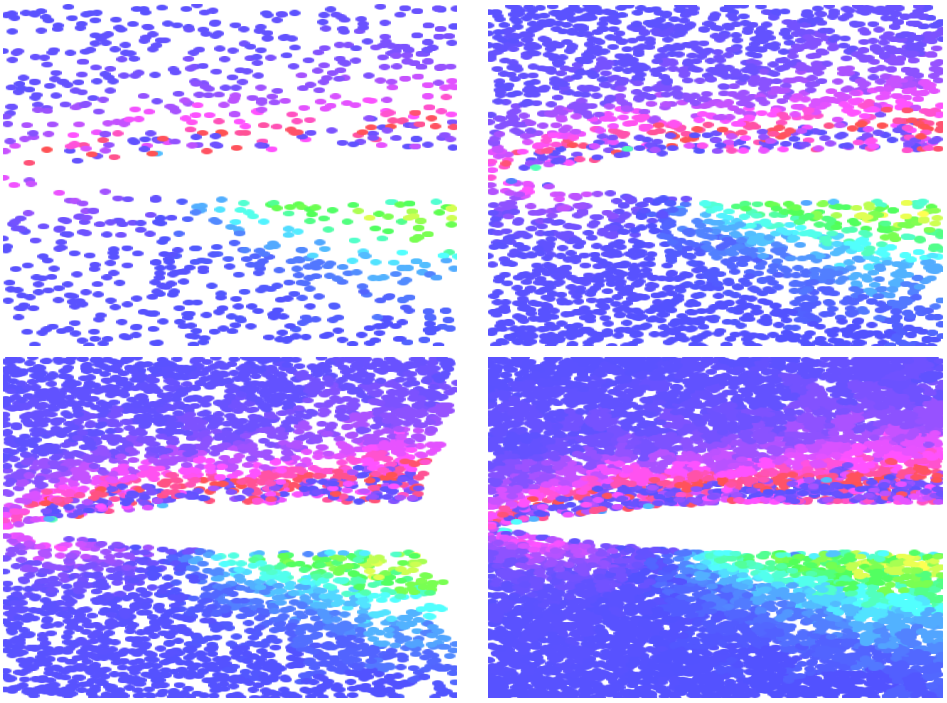


Figure 3.8: Random sampling with 1000, 2000, 3000 and 5000 points.

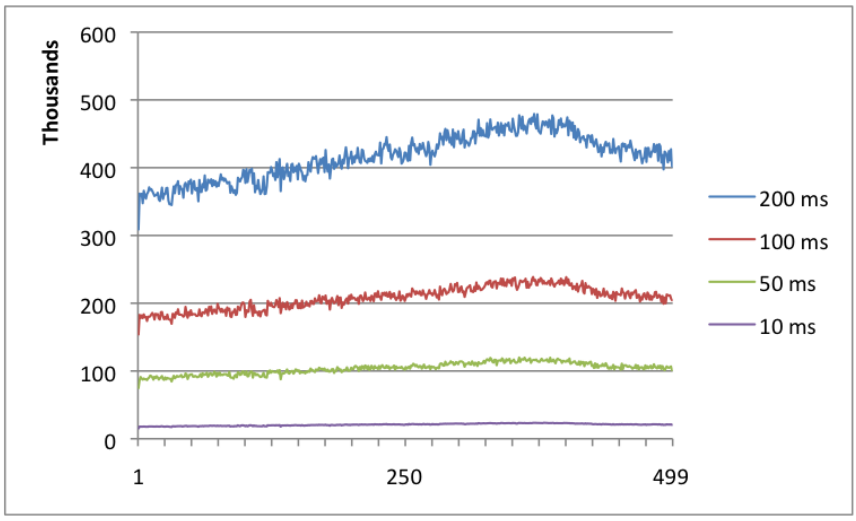


Figure 3.9: Number of sampling calculations performed within given time thresholds. While random sampling is was able to perform about 200,000 point samplings per 100 ms, progressive sampling achieved only a level with 65,536 points in the same time.

To determine the variance information, we apply a recursive calculation at initialization time. For each inner node $node_i$ in the cell tree, the variance v_i is determined as the sum of the child's variances

$$v_i = v_{n.left} + v_{n.right}.$$

The variance v_j of a leave node j is the volume-weighted sum of the variances v_c in each cell c

$$v_j = \sum_c v_c = \sum_c vol_c \cdot \sum_p (val_p - \hat{c})^2$$

where val_p is the point value at point p and $\hat{c}$ is the mean of the point values of the cell c. The extra amount of memory is small, since only one extra float value is stored for each inner search tree node.

Sampling positions and values are determined with following heuristic algorithm:

1. Set actual node n to the root node.
2. If node n is inner node
   (a) test if left and right child nodes n_l and n_r intersect with the cutting plane.
   (b) If none of n_l and n_r intersect, goto 4.
   (c) If only one of them intersects, set this one as node n. Goto 2.
   (d) If both intersect choose one randomly according to their variance and set this as node n. Goto 2.
3. If node n is leaf node
   (a) Test each cell inside leaf node in random order.
   (b) If cutting plane intersects with a cell, determine a position and value inside the cell. Set point radius to minimal extent of leaf node bounding box.
4. If less than n_{bound} points sampled, goto 1.

During gathering at the master node $points_{max}$ points have to be selected to be transferred to the front-end visualization. Therefore, following heuristic h is applied.

For each compute node a the sum of area covered by sample points is known, $area_p$, as well as the area of the intersection between the cutting plane and the compute node bounding box, $area_{BB}$. With num_p points we achieve constant average density with

$$h_a = min(area_p/num_p * area_{BB}, num_p)$$

sampling points inside this region. The total number of $points_{max}$ points is then selected by using n_a points per compute node a with

$$n_a = h_a / \sum h_a \cdot points_{max}.$$

Figure 3.10 shows four images demonstrating my adaptive point sampling scheme. The bandwidth parameter $points_{max}$ is chosen to be 1000, 2000, 3000 and 5000 sampling points.

Timing results for my adaptive sampling approach are shown in figure 3.11. This method is capable to perform even more sampling calculations per time compared to the random sampling approach. After 100 ms, my adaptive point sampling approach could extract up to 1.7 million points for some cutting plane positions. This performance boost is caused by the fact, that cutting plane intersection tests are used to find actual point positions (cf. step 2(a) and 3(b) in the algorithm). Even, if this method can determine more point information, this approach is more unbalanced than random point sampling. The highest performance is achieved for cutting plane positions intersecting many domain decomposition regions. In other regions, the performance is comparable to the one of the random point sampling strategy.

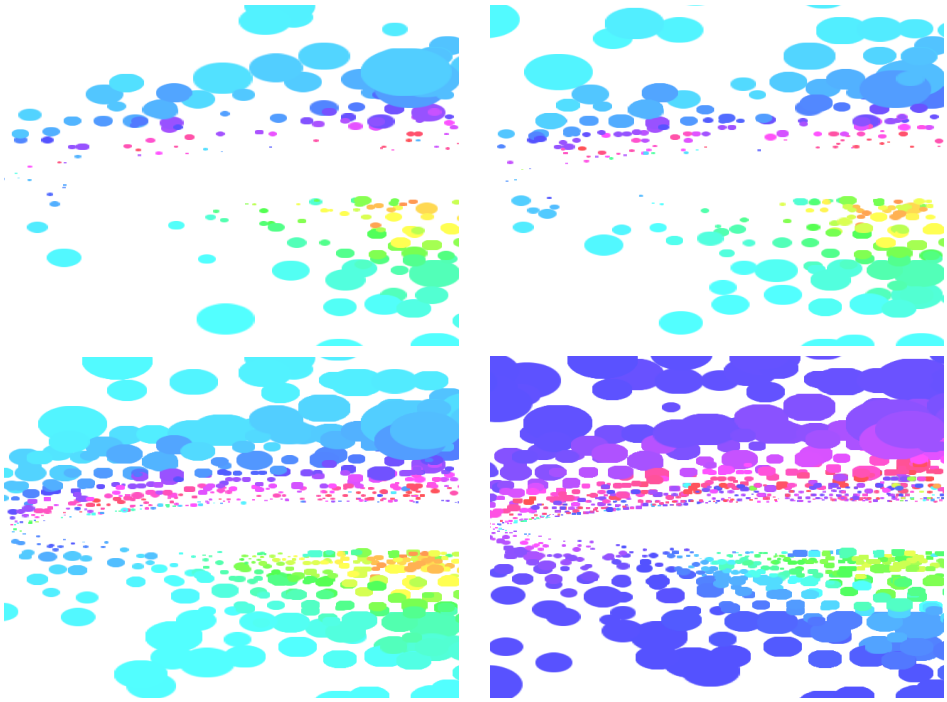


Figure 3.10: Adaptive sampling with 1000, 2000, 3000 and 5000 points, point size according to sampling region. Regions along the body are sampled more frequently, because of higher scalar field variance.

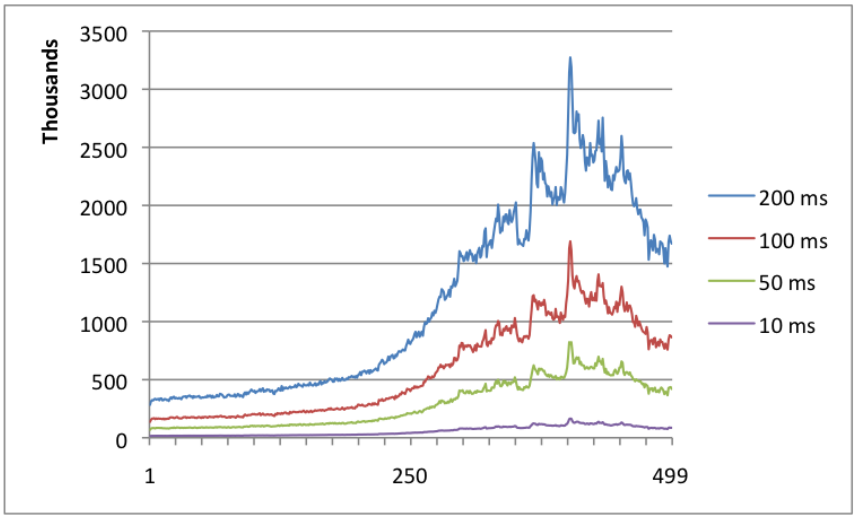
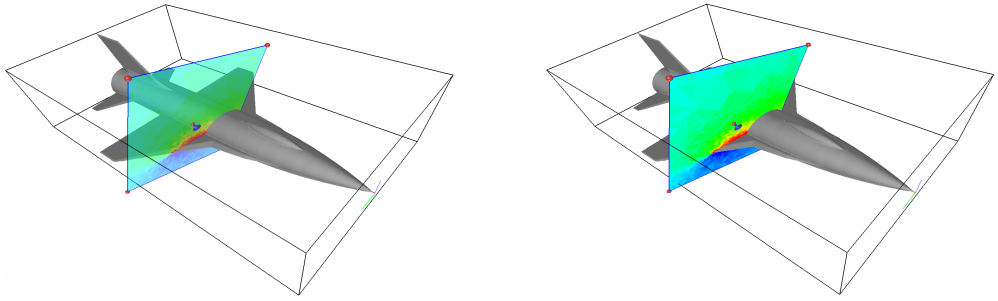


Figure 3.11: Number of sampling calculations performed within given time thresholds. In some regions, up to 1.7 million sampling points can be determined per 100 ms.

3.2.4 Rendering

In this section, I describe the interactive rendering process of extracted cutting planes on the virtual environment front end, which differs for the progressive sampling approach on the one hand, and random as well as adaptive sampling approaches on the other hand. Both rendering methods allow for interactive navigation in a virtual environment, see figure 3.12.



(a) Rendering a progressive texture (blended). (b) Rendering a point set (opaque).

Figure 3.12: Interactive rendering of in-situ extracted cutting planes: (a) progressive textures are reordered in the fragment shader, (b) random and adaptive point sets are rendered with approximated Voronoi regions.

For progressive textures I use an on-the-fly decompression. A natural method to visualize points sampled on a regular grid is the rendering of a textured quad. However, the ordering of the progressively sampled points is differs from the ordering that is assumed by the texture mapping units of graphics processing hardware. Therefore, the sampled points need to be reordered.

The final level, which will be extracted in the progressive sampling, is not known until the time threshold elapses. As a consequence, the reordering process needs to be performed for every progressive level arriving at the front end. Since a CPU-based solution is too computationally intense for the highly interactive scenario, a modified GPU fragment-shader is used to decode the z-curve on-the-fly during rendering.

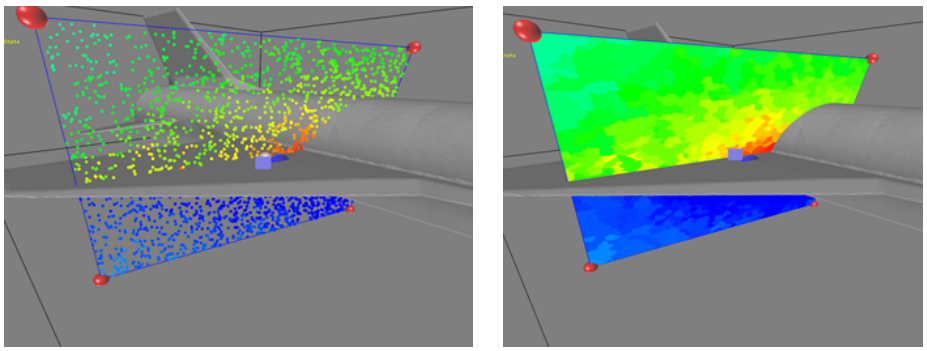
In order to render progressive sampled cutting planes, I implemented a look-up table approach, in which a current texture coordinate (x, y) is mapped to the level l and

coordinate $(\hat{x}, \hat{y})$ of a z-order curve. During initialization, these look-up tables are pre-generated and stored in textures for each possible z-curve pattern. For a texture coordinate $(x, y) \in [0 \dots 2^n - 1, 0 \dots 2^n - 1]$ the level l and local coordinates $(\hat{x}, \hat{y})$ can be determined with the minimum number of trailing zero bits z of x and y by:

$$\begin{aligned} l &= n - z, \\ (\hat{x}, \hat{y}) &= (x >> z, y >> z). \end{aligned}$$

In order to render the point sets generated by the random or adaptive point sampling approaches, a visual pleasing representation is desirable. Since a finite set of points in a plane is given, an obvious solution is to fill the plane with nearest neighbor information. Therefore, a tessellation of the cutting plane is required, in which each cell consists of every point whose distance is less or equal to any other point in the point set. This, by definition, is the Voronoi diagram of the point set.

A real-time generation of Voronoi diagrams is still a challenging task. Therefore, an approximation to the Voronoi diagram based on jump flooding is used capable to be performed in a constant time on graphics processing units [RT06]. With a fixed texture resolution of 1024 by 1024 texels, result shown in figure 3.13, I was able to provide a local frame rate of about 180 frames per second.



(a) Point set. (b) Approximated Voronoi regions.

Figure 3.13: Real-time rendering of point sets with approximated Voronoi regions: (a) original point set, (b) rendered as Voronoi texture.

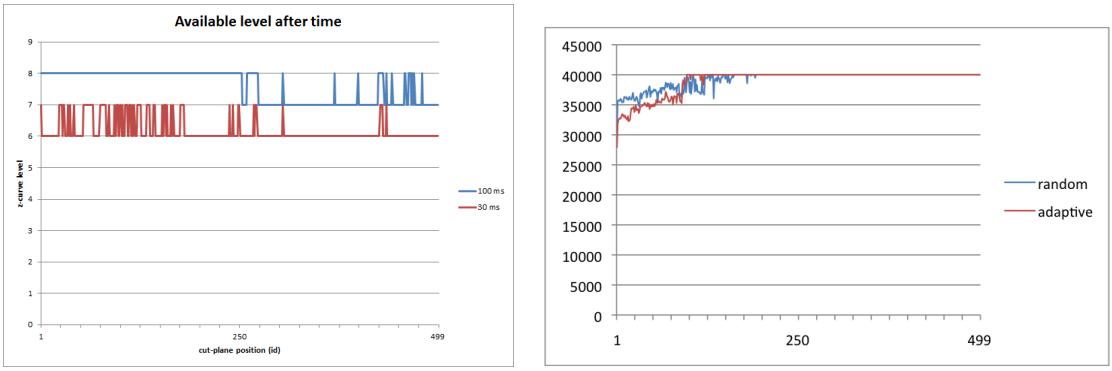
3.3 Results

In this section, I will discuss my achieved results. I will demonstrate the great benefits of my in-situ sampling approaches which enable for using virtual environments for online monitoring purposes of running large scale simulations. This enhances the online monitoring process to benefit from high interactivity and immersion provided by virtual reality techniques.

In order to interactively explore the state of an online simulation, certain constraints are required on the extraction algorithms and update rates. Therefore, I presume that extraction run time can be bounded in order to stay interactive. By using the presented progressive sampling, the extraction algorithm can easily be stopped after each refinement step. On the other hand, even when stopped early, intermediate results should support an overview inside the current cutting plane domain to enable the movement of the cutting plane in an explorative way. Both requirements can not be achieved by cell-based extraction algorithms. However, the progressive sampling approach met both requirements.

Figure 3.14(a) demonstrates the effect of introducing time constraints on the progressive sampling approach. When using different time thresholds, a varying number of progressive levels could be transferred to the front end in the given time. This allows requesting results for new cutting plane positions while moving the cutting plane and render preliminary results. In figure 3.14(b), the number of points, which could be transferred within a time threshold of 100 ms is shown. In this benchmark, a 100-MBit/s-Ethernet connection was used. Even if more point samples could be achieved during this period, bandwidth limitations prevented to render a more detailed visualization. Here, a different point encoding and compression could result in better results.

Using virtual reality techniques for online monitoring leads to beneficial results. I demonstrate these benefits by coupling a parallel CFD simulation to a powerwall system equipped with a FlyStick interaction device. This clearly enhanced the online monitoring in many ways. Stereoscopic rendering improved grasping the geometric relationships between context geometry and the information presented on the cutting plane. Furthermore, virtual environments provide better interaction techniques. I



(a) Progressive texture levels.

(b) Transferred points.

Figure 3.14: Visual quality of interactive cutting plane approaches. Time constraints result in limited details: (a) number of progressive texture levels, (b) number of transferred points per 100 ms.

rendered a cutting plane widget representing the cutting plane edges. This widget can be picked intuitively in order to change the cutting plane position as well as its extends (see figure 1).

3.4 Conclusion and Future Work

In this chapter, I showed that virtual environments can be successfully used in order to interactively explore the state of a running simulation. Therefore, I combined intuitive manipulation of cutting plane positions with in-situ cutting plane extraction. Since cell-based algorithms can not fulfill the requirements of the interactive frontend, I used different point sampling approaches.

Future work would incorporate even more intuitive interaction metaphors. Figure 3.15 shows a possible solution using a pad device allowing for cutting plane interaction as a tangible interaction device. Furthermore, two-hand interaction used for cutting plane interactions would also be an interesting research topic.

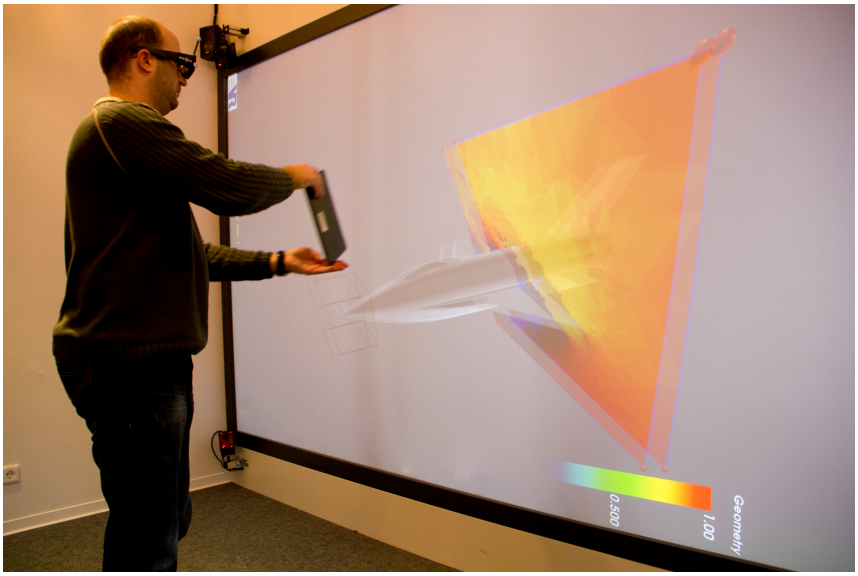


Figure 3.15: To further improve intuitive interactions, different interaction metaphors could be introduced, such as using pad devices as cutting plane surrogates.

Chapter 4

Interactive Hybrid Rendering in Virtual Environments

In this chapter, I deal with the interactive visualization of large-scale scalar fields incorporating remote rendering techniques.

Increasing capabilities of modern high-performance computing resources lead to the simulation of complex physics substituting a large amount of physical experiments. While the compute power provided by supercomputers and large-scale cluster systems enabled for simulating more complex numerical equations, simultaneously, spatial and temporal resolution is growing. For a comprehensive understanding of the numerical simulation data, confirmative analysis is often not sufficient and scientists need to study their results in an explorative way.

In virtual environments, explorative analysis benefits from high immersion giving better insights into the simulation data. Furthermore, hypothesis can be justified and altered with more natural interaction techniques. Nevertheless, interactivity is crucial for this kind of explorative analysis. Therefore, high update rates and low latency are required. However, large-scale simulation data is growing and the extracted post-processing data is overwhelming the available graphics hardware driving the virtual environment systems. A solution to deal with this massive data sets is the usage of parallel remote rendering. However, major limitation of remote rendering approaches is high latency.

In this chapter, I examined the application of hybrid rendering techniques in order to interactively navigate through large-scale simulation data, a process that can be applied for online monitoring purposes or to analyze final post-processed simulations. First, I give an overview of state-of-the-art techniques for hybrid rendering and work related to this chapter in section 4.1.

Section 4.2 describes my hybrid rendering technique which combines the benefits of local and remote resources. The rendering workload of the visualization is divided into parts for local and remote rendering. While geometric components are rendered locally and provide interactive navigation, complex flow features such as iso-surfaces are rendered in parallel remotely. Due to tracked viewers and network latency, locally and remotely rendered images tend to diverge in time and space and need to be re-adjusted. I present an extensive evaluation of my presented approach utilizing a multi-pipe system which demonstrates the benefits of my enhanced rendering approach.

In section 4.3, I focus on the usage of display walls predestined for visualizing high-resolution data. An important application scenario is on-line monitoring of large-scale simulations, in which copying raw data becomes a severe bottleneck. Instead, in-situ rendering is reducing this overhead by copying visualization results. Here, I present a progressive image streaming which can handle high latency caused by large pixel counts and network bandwidth limitations.

Finally, I discuss time-dependent data sets in section 4.4. Here, a standalone interactive exploration workstation suffers from exceeding local memory capacities or insufficient disc I/O performance easily. This data-handling task is burdened to a dedicated GPU cluster system. I exploit remote GPU hardware for rendering in addition to interactive iso-surface extraction in order to provide interactivity.

4.1 State-of-the-Art and Related Work

4.1.1 Remote and Hybrid Rendering

Remote or distributed rendering techniques for interactive high quality scientific visualization of three-dimensional data sets are of high importance in modern times characterized by mobile devices and constant hardware change. Most of these approaches are based on client/server architectures [SHA+09] over network to bypass scenarios where the data to be rendered is distributed across different resources, possibly located at remote locations. Remote rendering systems can be classified through the decision which steps of the visualization pipeline run on the client side or the server side [SHA+09] [SG96] (see figure 4.1).

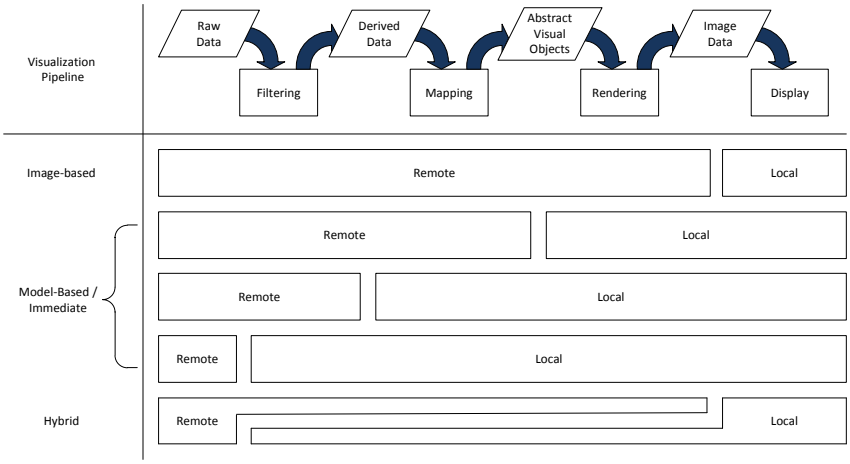


Figure 4.1: Classification of remote rendering techniques by their distribution of the pipeline. While image based methods render remote, model based techniques use local resources for rendering. Hybrid methods share the rendering effort.

A common approach is image based rendering (IBR) where the rendering is performed on the server side and only pixel data is transferred back to the client [Pro08]. This pixel stream can also include further information like depth or accumulation buffers. The main problem using image based rendering methods is the interaction delay. This delay is described by the waiting time from requesting new images to

their arrival. This delay is especially unacceptable when using VR environments where the user's head position is continually changing.

Model-based rendering (MBR) approaches are more friendly to latency because the rendering is performed locally and therefore geometry or raw data from the server is transferred to the client. However, this approach results in high bandwidth requirements for the geometry transfer. This method also requires that the visualization frontend is powerful enough to handle the incoming geometry data for rendering. Common turnkey applications in the field of scientific visualization like ParaView [LHA01] or Visit [VIS05] are supporting model- and image-based rendering.

Another more generic approach called immediate-mode, intercepts low-level drawing commands on the client and sends them over network to the remote server for rendering. Many frameworks, such as ClusterGL [NHM11], Chromium [HHN+02] and WireGL [HEB+01], use this mode for a generic solution to enable default OpenGL applications for remote rendering. Most of these frameworks are also suitable for tiled display configurations. Nevertheless, these frameworks also have to tackle with delays and reduced interactivity due to high network overhead.

Hybrid rendering approaches therefore try to combine image- and model-based rendering to exploit the advantages and reduce the disadvantages of these techniques [EHT+00] [NSOJA11] [SLRF08]. This improve interactivity and reduce bandwidth by using remote and local resources for rendering. In most cases parts of the geometry are rendered with low-resolution and high frame rates by the client itself. The server only transmits images and corresponding depth values per pixel from highly complex scenes to the local client, which then composite these two images to a final result [ADD+07].

An example using combined local and remote visualization techniques for interactive volume rendering in medical applications is described in [EEH+00]. The direct rendering of the volume is done with 3D texture mapping methods on specialized remote hardware while the client is responsible for rendering 2D orthogonal slices of the volume. Noguera et al. used hybrid rendering techniques to navigate in large terrains using mobile devices. The terrain area close to the viewer is rendered in

real-time by the mobile client and the terrain area located far from the viewer is rendered as a panorama image on the remote host [NSOJA10].

The presented hybrid rendering approach, different from the above applications, also makes use of 3D image warping [CW93] [MB95] [MMB97] [HM99], aiming to hide the delay through re-positioning of the previous remote image to the actual position when the correct image is not available.

4.1.2 Parallel Rendering

When the number of polygons or fill rate exceeds the compute or memory capacity of single graphics cards, parallel rendering techniques have proven to be a useful solution. A first classification of these parallel rendering techniques is described in [MCEF08]. The common parallel rendering framework IceT [MWP01], also used in this section, is based on sort-last image compositing, which seems to be a scalable approach for parallel rendering. Another framework for the development of parallel rendering applications is Equalizer [EMP09]. This framework also supports sort-first parallel rendering but is a little bit overloaded to our needs.

4.2 Hybrid Rendering System for Interactive Navigation

Large-Scale numerical simulations produce complex data sets desirable to be explored interactively in virtual environments. However, interactivity is asking for enhanced methods in order to process and render large-scale simulation results at high frame rates. On the one hand, such data sets can easily exceed tera- or petabytes, moreover exa-scale simulations are expected in the near future. On the other hand, typical virtual reality hardware has limited resources. In order to achieve high frame rates, rendering has to utilize the acceleration provided by modern GPUs which have limited amount of video RAM and render performance. For instance, NVIDIA's stereo-capable Quadro FX5800 graphics card has a maximum specified triangle throughput of 300 million triangles per second. This implies a maximum

amount of 5 million triangles render-able at a recommended frame rate of 30 frames per second in full-stereo.

In this section, I introduce a software architecture that offers a hybrid rendering approach in order to overcome the gap between available render performance and the size of large-scale data sets. My presented strategy divides the rendering workload of the visualization into parts assigned to local and remote rendering resources which enables for supporting interactive navigation through a data set in virtual environments. While geometric components are animated in a local virtual environment, rendering of complex flow features such as iso-surfaces is performed in parallel on a remote GPU cluster. The images and depth values of the remote images are then streamed back to the local virtual environment. Here, local context geometry is rendered with high frame rates and local and remote images are composed again. Due to viewer tracking and network latency, locally and remotely rendered images tend to diverge. Therefore, both images are re-adjusted utilizing a point-based rendering solution. I implemented and tested the presented system on a three-pipe powerwall system visible in figure 4.2.

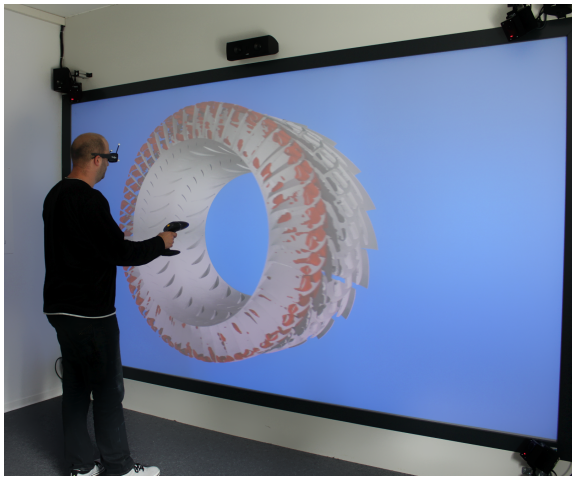


Figure 4.2: Interactive hybrid rendering at powerwall of German Aerospace Center: Context geometry (rotor blades, gray) is rendered locally, iso-surface is rendered remotely on a GPU cluster and combined into the scene.

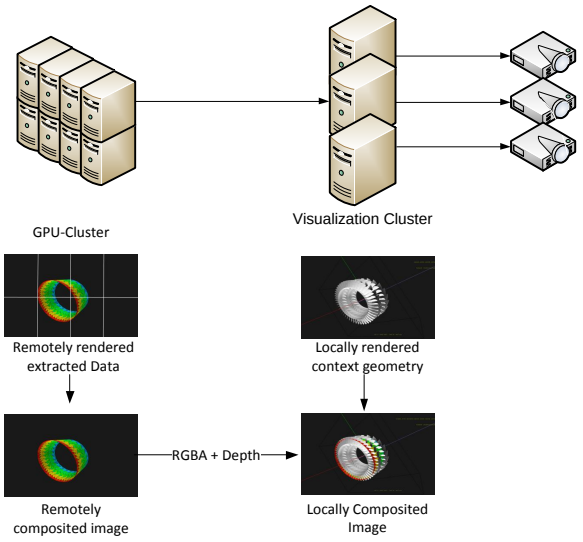


Figure 4.3: Hybrid rendering approach: Remote rendered images are composed on a GPU cluster and transferred to the frontend. Here, they are combined with the local context geometry according to the pixels depth values.

4.2.1 Hybrid Rendering Framework

This section presents the details of my hybrid rendering architecture used for interactively navigate through large-scale data sets.

4.2.1.1 System Design

My parallel rendering framework, as depicted in figure 4.3, is based on a client/server architecture and consists of a local frontend application and a parallel remote rendering application, both based on the ViSTA [AK08] and ViSTA FlowLib frameworks. The frontend application is running on the visualization cluster driving, for instance, a powerwall and the backend application is running on a GPU-Cluster used for the generation of remote images.

To be scalable with increasing data sizes, my approach is based on a sort-last image compositing, utilizing the IceT image compositing framework. This is expected to be the only scalable approach for rendering large-scale data sets, balancing the rendering workload and memory requirements over the GPUs. Therefore, rendering

primitives from the extracted features are fairly distributed across all rendering processes to guarantee high or constant frame rates.

My developed frontend application is also running in parallel, mirroring the application state to each of its processes. One instance of this distributed application runs as master and all other processes as slaves. In this configuration, the master is responsible for frame synchronization and for user interaction. All interaction events are distributed from the master to the slave nodes for correct rendering of the local context geometry. This context geometry is rendered by every frontend instance. The master node also establishes a TCP/IP connection to the parallel renderer for requesting and receiving images generated by the remote application.

4.2.1.2 Hybrid Rendering

In order to enable for interactive navigation, I make use of hybrid rendering. Therefore, a remote image is requested from the parallel renderer and combined with the local rendering.

When the current camera position or orientation is changed by user movements, a render request is send to the remote application. This request is including the current model view matrix and further meta data, e.g. compression parameters. A new image is then generated in parallel. When the IceT rendering process is finished, each remote tile compresses its color and depth buffer in parallel using OpenMP and sends them to the master node of the backend application. The color and depth buffers can be compressed with different compression methods. These are standard ZLIB, LZO and JPEG compression. JPEG compression is optimized by using libjpeg-turbo, which is a faster derivative of libjpeg. When all buffers are collected by the backend master, a message including all buffers from each tile is send to the frontend application master.

On the frontend side, the master node is then distributing the remote image information across all slave nodes. After each slave has received the message, all buffers are decompressed in parallel using OpenMP. Finally the remote image buffers are composed with the locally rendered image by depth value comparison of each pixel. This step is done on the GPU using the OpenGL Shading Language (GLSL).

4.2.1.3 Image Adjustments

Combining local and remote rendered images without adaptation leads to image artifacts. Since locally rendered context geometry is moving continuously and remotely rendered images are delayed, the features displayed on both images usually do not fit. Figure 4.4(a) and figure 4.4(b) demonstrate this effect. Figure 4.4(a) depicts a situation where both images do match and the camera is rotating around the scene in figure 4.4(b) so that the images are diverged. Obviously, a simple combination of both images according to depth values is not sufficient. Instead, I use depth-image-based rendering as described in section 4.1.1. For this reason, I first treat the remote image pixels as points in space according to the depth textures.

To transform the pixels to their new positions fitting to the current camera my framework applies the following transformations. An object-space point p in the remote image was transformed with the remote model view projection matrix MVP_{remote} followed by perspective division $div(p)$ and the viewport transformation VP_{remote}:

$$p' = T_{remote}(p) = VP_{remote} \cdot div(MVP_{remote} \cdot p).$$

This transformation can be inversed with the inverse mapping:

$$p = p'' = T^{-1}_{remote}(p) = div(MVP^{-1}_{remote} \cdot (VP^{-1}_{remote} \cdot p'))$$

The point's new position in the current rendered image can then be calculated by

$$p''' = T_{local}(p'') = VP_{local} \cdot div(MVP_{local} \cdot p'').$$

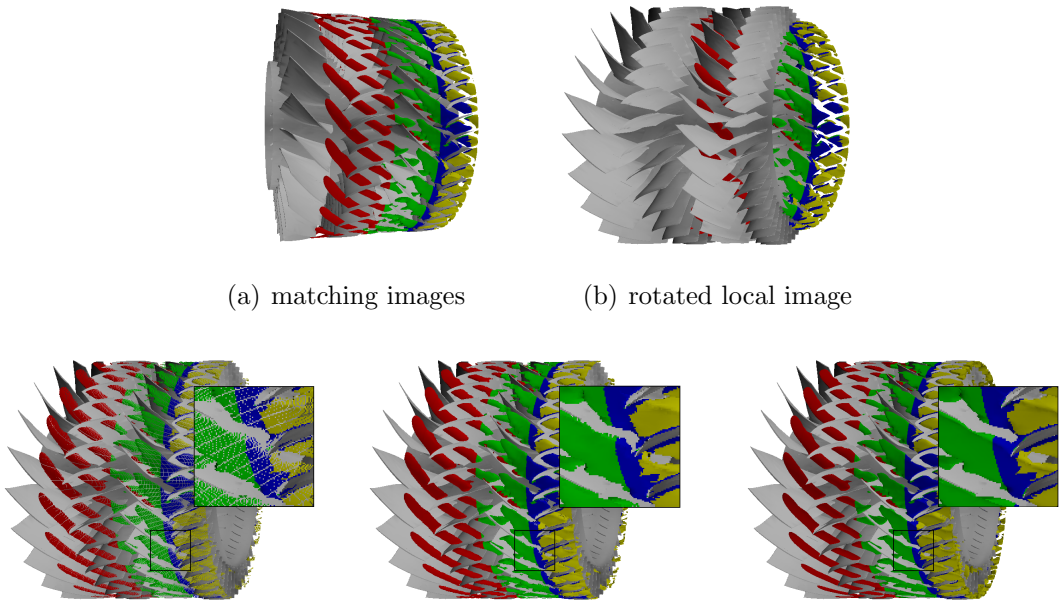
In summary, the transformation of points p' determined from the remote images to vertices p''' in the current view can be stated as

$$p''' = T(p') = T_{local}(T^{-1}_{remote}(p')).$$

All involved matrices are uniform while rendering the current local image. The transformation $T(p')$ from remote image pixels to the new vertex positions is applied

efficiently inside the vertex shader of a local GPU. In order to draw a vertex for each screen pixel, static vertex buffer objects (VBO) are used. For each vertex, its depth- and color-value is read from textures. Figure 4.8(a) shows an example of this transformation rendered with a point for each transformed vertex.

Since my presented framework distributes remote images to each frontend client, every rendering process has all buffers from each tile available. Therefore, the adaptation is also possible over tile transitions which fills gaps in the rendering of distinct viewports, which are present in tiled-display or multi-pipe powerwall configurations. In addition, this technique generates the required image pairs for rendering in stereo scenarios. This has the advantage that there is no need to generate and transfer images for the left and the right eye, reducing the remote render overhead by half.



(a) matching images (b) rotated local image

(c) fixed point-size adjustment (d) adaptive point-size adjustment (e) correct remote image

Figure 4.4: Exaggerated hybrid rendering example. The upper left image is showing a correct compositing of a matching remote and local image. In the upper right image the user is rotating the scene so that the local and remote image do not fit anymore. The lower left image shows the re-adjusted image with simple point based rendering and in the lower middle with adaptive point-sizes to fill holes. On the lower right image correct rendering is shown.

However, this approach generates visual artifacts, compared to the correct rendered image presented in figure 4.8(c), because the transformed images do not provide information for all visible pixels. While advanced hole and occlusion filling approaches exist, nevertheless, they all have problems dealing with background pixels. Pixel-based methods suffer from rubber band and blending effects, geometry-based methods require for the definition of depth thresholds in order to distinguish between foreground and background pixels. This definition is very sensitive and can either result in wrong connected regions or in not filling all holes in surfaces. In addition, they are computational intense and do often not fulfill the rendering requirements of virtual environments. A third class of methods is the combination of multiple viewports. For these methods the number of required viewports is often unclear and they have high rendering requirements as well.

Due to these limitations of occlusion filling strategies, my presented image adaptation deals with the artifacts by only filling the appearing holes in rendered surfaces without aiming on filling formerly occluded image parts. Therefore, adaptive point size adjustments is applied and the result is shown in figure 4.8(b). The required point size is determined during point transformations in the vertex shader and is depending on the derivation in x- and y-axis. Therefore, the screen-space distance to the transformed neighbor pixels with same z-values is calculated and used as point size in the following OpenGL pipeline. This point-based rendering approach successfully fills the holes inside a surface in the remote image.

4.2.2 Results

The clear benefits of the presented hybrid rendering approach were evaluated in a powerwall scenario which I present in this section.

4.2.2.1 Data Set

To benchmark my approach a computational fluid dynamics simulation of the RIG250 compressor turbo-machinery is used, provided by the Institute of Propulsion Technology at the German Aerospace Center in Cologne.

The simulation utilizes a multi-block structured grid which consists of 123 million mesh points. From the simulations' fluid motion field an iso-surface is pre-computed. This is rendered on the backend running on the remote GPU-cluster. The isofurface consists of 83.4 million triangles and can not be rendered in real time in the virtual environment.

The local frontend renders a static geometry of the RIG250 compressors' surface in order to visualize an interactive navigation context. This surface consists of two rotors and two stators with 4.5 million triangles and can be rendered in real time on the frontend.

4.2.2.2 Hardware Configuration

The hardware infrastructure involved in the presented benchmarks is a parallel GPU cluster connected to a powerwall via 1 Gbit Ethernet.

The frontend application is running on an available powerwall VR-environment with three pipes. Each of the three screens has a resolution of 1050 by 1400 pixels, since the projectors are rotated by 90 degrees to have a high vertical and horizontal resolution. One screen is connected to one machine for rendering. These machines are equipped with two Intel Xeon X5560 quad-core processors with 24GB DDR3 main memory and one NVIDIA Quadro FX 5800 graphics card with 4GB DDR5 graphics memory.

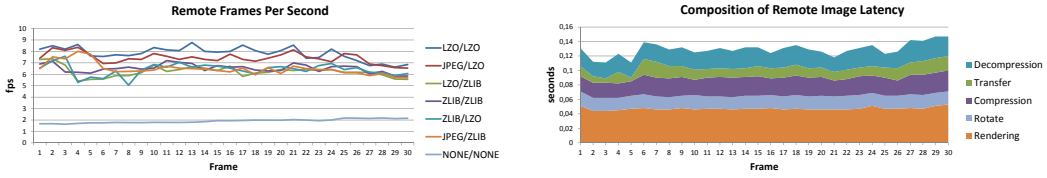
The GPU-Cluster running the parallel rendering application consists of four high-end visualization workstations. Each of these workstations has two Intel Xeon X5670 hexa-core processors, 48GB of DDR3 main memory and three NVIDIA Quadro 6000 graphics cards with 6GB DDR5 graphics memory. The local interconnect for these machines is a 40Gbit QDR Infiniband network. For the benchmarks we used three nodes with only one GPU per node to generate the remote image.

4.2.2.3 Benchmarks

In order to define a repeatable benchmark suite, I execute all the following benchmarks in the same scenario, in which the complete scene to render is rotated by a constant angle per rendered remote frame.

First of all, it is important to note that the local frame rate was always about 45 frames per second in a full-stereo rendering. This means that interactivity was always provided on the frontend.

When working with interactive visualization, latency is the most important property. Therefore the time from requesting a remote image until it has been distributed and decompressed and is therefore available for rendering on all tiles was measured. This benchmark has been executed with all combinations of compression algorithms. Figure 4.5(a) shows the achieved remote frame rates with all combinations of the compression algorithms.



(a) (b)

Figure 4.5: Performance and latencies of the remote rendering benchmark. (a) Remote frame rates with different combinations of image and depth buffer compressions. The used compression is titled as 'Image Compression' / 'Depth Compression'. (b) Composition of the produced latency when using JPEG for color compression and ZLIB for depth compression.

When using the LZO compression for depth and color values, a remote frame rate of about 8.0 frames per second was achieved. This means roughly four frames on the frontend are the same and need re-adjustments when rendering at 30 frames per second until a new remote image is available. With this remote update rate the user can merely see artifacts because the error produced by the local re-adjustment is small. However, when using LZO for depth and color buffer compression, the required bandwdith is high. Therefore, for lower bandwidth network connections color compression via JPEG and depth compression via LZO is a more suitable

approach which achieved in average 7.5 frames per second, here. Using ZLIB as compression for both or only of the buffers only 6 frames could be archieved. This is due to slow ZLIB compression and decompression. With uncompressed buffers only 2 frames could be achieved due to larger image sizes which leads to more visual artifacts.

Figure 4.5(b) shows a composition of the latency when using JPEG for color encoding and LZO for depth compression. The most time-consuming part is the IceT parallel rendering with 47ms in average. Compression and decompression could be parallelized efficiently. All six buffers (3x depth buffers and 3x RGBA buffers) can be compressed in 25ms and decompressed in another 25ms. Since my visualization frontend is working with rotated projectors the remote images have to be rotated as well which takes 17ms in average. Finally, the image transfer itself needs about 13ms. In total, a delay of 127ms is achieved for an update of the powerwall.

Another important fact when using remote rendering is the required bandwidth. Therefore, I measuered the size of all frame buffers. Figure 4.6 shows the compressed sizes for different combinations of compression codecs. An uncompressed frame buffer for the entire powerwall takes 34.4 MB per frame. The highest compression rate is achieved using JPEG for color compression and ZLIB for depth compression. However, using this codec combination is more time consuming due to the slow ZLIB compression. Using LZO the fastest compression method produces an average framebuffer size of 1.8 MB which is small enough for fast image streaming over 1 GBit network connections.

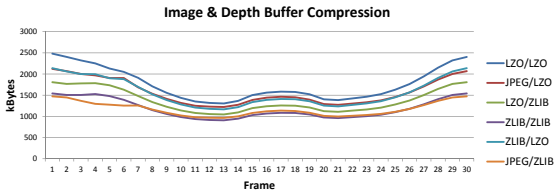
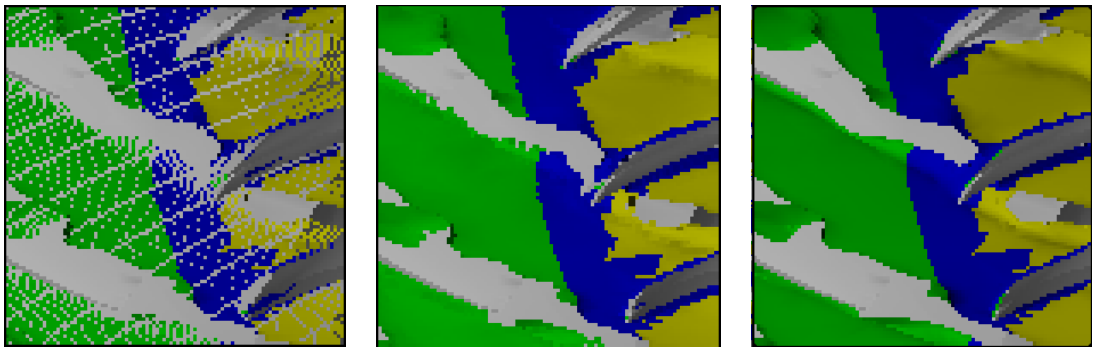


Figure 4.6: Size of the transferred framebuffers (color + depth) using different combinations of compression codecs.

4.2.2.4 Image Quality

Point-based rendering with adaptive point sizes, as used in my presented framework, successfully fills the holes inside a surface in adapted remote image. Nevertheless, boundaries tend to be a bit unsharp (compare the renderings in figure 4.7(b) and figure 4.7(c)).



(a) fixed point-size adjustment (b) adaptive point-size adjustment (c) correct remote image

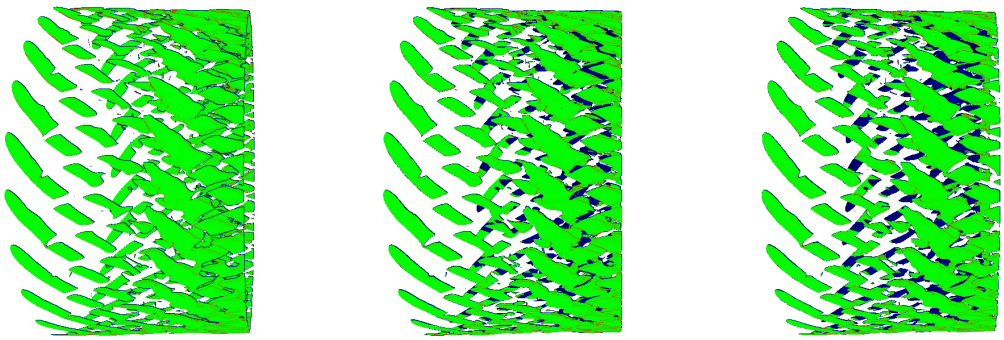
Figure 4.7: Image quality in the hybrid rendering framework: depth based rendering without (a) and with (b) adaptive point sizes as well as a reference image (c).

Background filling of previously occluded image regions is not applied in my framework, therefore, previously occluded parts of surfaces are also missing in the adapted image. This is demonstrated in the image sequence depicted in figure 4.8, showing missing surface regions in blue.

Background filling methods try to fill these missing informations based on the surrounding available image pixels. For instance, [BSF10] uses lines to fill background information at detected edges. The problem with these methods is that threshold needs to be defined to detect edges by different z-values. Therefore, components might be wrongly connected or foreground surfaces are extended into the background. More important, complex hidden surfaces in the background can not be extended correctly.

In conclusion, my presented framework is not filling the background, because missing information caused by occluded regions is often a better solution than rendering probably suspicious information. For a user it should be easier to deal with missing

information that will be filled in the next frames rather than to decide when surfaces turn from being extended to being correctly rendered. Furthermore, as shown in the benchmark section 4.2.2.3, the remote image was updated every 4 local frames in average, minimizing the area in which background filling is required. Besides, in the compressor benchmark scenario most of these background areas are occluded by context geometry anyways.



(a) Updated position (b) rotation by 2.5 degrees (c) rotation by 5 degrees

Figure 4.8: Missing information without background filling from different viewpoint angles: (a) updated remote image, rotation by 2.5 (b) and 5 (c) degrees. Occluded surface regions that would be missing in adapted images are shown in blue.

4.2.3 Discussion

My hybrid rendering framework presented in this section provides a technique to interactively explore simulation data sets with high spatial resolution. This is important for interactive navigation in large-scale data sets, as it prevents virtual reality hardware and network connections from being performance bottlenecks.

My approach divides the rendering workload into rendering of geometric context components and rendering complex flow features, such as iso-surface visualizations. While geometric context components can be rendered locally with high frame rates, complex features are decoupled and rendered remotely. Remote rendered images have lower update rates and arrive with latency, which leads to mismatching visual information in reference to the current local rendering. Even, when using lower-

resolution remote images, latency is inherently present due to rendering, compression, and network transfer times.

My presented system handles these divergences by re-adjusting remote rendered images with depth image based rendering and adaptive point sizes. It was demonstrated, that hybrid rendering is capable to support interactive navigation through a data set and still provides high frame rates even if remote images can only be rendered at lower frame rates.

Future work should focus on better support for transparent images as well as image quality. Due to the division into images rendered remotely and locally, transparent pixels are rendered correct only if the z-order of remote and local triangles do fit. Incorporating multi-pass rendering could be a strong improvement, nevertheless, introducing additional rendering workload and requiring for the transfer of multiple frame buffers. Since my framework adapts only pixel positions without re-shading them, their color differs slightly from the correctly rendered image. The usage of deferred shading, which involved transmission of normal buffers, could be a solution, here.

4.3 Streaming Large Images for Tiled Display Walls

In section 4.2 I described my method to provide interactive navigation of large-scale data sets in virtual environments, even if their size exceeds the rendering capabilities of the available hardware resources. My presented hybrid rendering technique divides rendering primitives in portions rendered locally or remotely. In this section, I discuss the challenges and give a possible solution burdened by the network bottleneck are discussed which arise with increasing display resolutions.

Complex simulations often require online monitoring, the visualization of the running simulation, to understand the simulation behavior. Here, explorative visualization supports a quick overview to identify interesting regions. A common solution is to copy raw simulation data from the supercomputer to a dedicated visualization cluster system. This visualization cluster performs post-processing algorithms and extracts features to be visualized. In the end, rendered images of these features are finally shown on a frontend system.

While future high performance computing (HPC) resources will dramatically improve compute power, I/O and network bandwidth are expected to grow to a lesser extent. Thus, copying raw data becomes a severe bottleneck due to long transmission times. In-Situ processing is addressing this problem by shifting post-processing extraction algorithms to the compute resources already used for simulation tasks. Instead of raw data, intermediate visualization results can be copied, which are normally decreased by orders of magnitude.

Simultaneously, with enhanced HPC capabilities simulation details and spatial resolution will grow. This also requires high resolution displays for the visual perception of fine details. While classical display systems or virtual reality environments do not provide sufficient resolutions in all application cases, display wall systems have proven to support very high resolutions up to tens or hundreds of megapixels.

Nowadays, these display wall systems are driven by a set of local computers which render the complete information or display images that are rendered on remote rendering clusters. On the one hand, local rendering solutions support low latencies

of the rendered information. But, with increasing intermediate visualization results insufficient render performance or memory is locally available. On the other hand, parallel remote render solutions benefit from huge distributed memory amounts and render performance. However, they have higher latencies and with limited network bandwidth traditional renderings require high subsampling rates in order to achieve acceptable update rates.

Figure 4.9 describes our target hardware infrastructure. Features extracted by the post processing step, which is running on the same resource (supercomputer) as the simulation, are streamed to a remote rendering application via a high speed network. The data size transferred to the remote GPU cluster is much smaller as the raw data produced by the simulation but can also be to large for the visualization cluster driving the display wall. To balance the rendering workload over the GPUs of the GPU cluster, the extracted features must be fairly distributed across all rendering processes to guarantee high or constant frame rates. Furthermore, spatial domain decomposition of the rendered data becomes an important task. Because data needs to be rendered at least once for every tile it is visible on, data covering all tiles lead to unbalanced or multiple render passes.

My solution to enable for interactive exploration of online monitored simulations using tiled display walls is presented here. Here, I simulate the online monitoring process by loading already extracted isosurfaces at the remote rendering side instead of performing in-situ processing. This is a viable solution, since I am focusing on the interaction between parallel remote rendering and tiled display wall frontends in this section. In doing so, render solutions for future architectures, which are not available right now, can be defined and evaluated.

My presented hybrid rendering framework presented before, combining local rendering resources used for context geometry and remote rendering techniques rendering complex geometry, is extended here. Therefore, I introduce progressive and multi-resolution image streaming based on z-order curves into the system. Moreover, in order to concurrently stream rendered images independent of the remote render process, I extend my hybrid rendering framework with pipelined processing steps.

In summary, my approach achieves the following benefits:

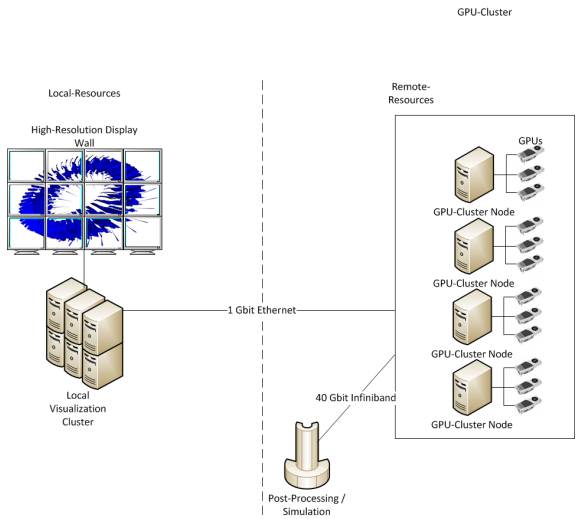


Figure 4.9: Hardware Infrastructure: A visualization frontend running on a local visualization cluster, which is driving a display wall, is connected via gigabit Ethernet to a remote GPU cluster equipped with multiple GPUs. Extracted features of the post-processing / simulation are transferred via a high speed Infiniband network to the GPU cluster for rendering.

- The hybrid rendering using local and remote resources allows interactive navigation through the dataset.
- Multi-resolution, based on z-order curves, supports quick overviews.
- The progressive image streaming introduces only low latencies to the remote rendering process.
- Using progressives streaming instead of subsampling requires no additional render passes for different resolutions and adapts automatically to network capabilities.

4.3.1 Modified Framework Architecture

The basic design of my system presented here is the same as I described in the last section (see section 4.2.1.1, section 4.2.1.2, and section 4.2.1.3). However, if the rendering process finishes a remote image, it is passed to a concurrent thread for transmission, which creates the progressive data structure, compresses each data

packages via standard zlib compression and transmits the results back to the frontend. The progressive streaming itself is described in detail in section 4.3.2.

The fact that I handle the output connection by an additional thread, allows for a pipelined execution (see figure 4.10) of rendering and transmission. This exploits today's and future multi-core hardware architectures via multi-threading. This thread is also streaming the previous image to the frontend until a newer image is available for transmission. This means that the previous frontend image is getting finer until a new image was rendered.

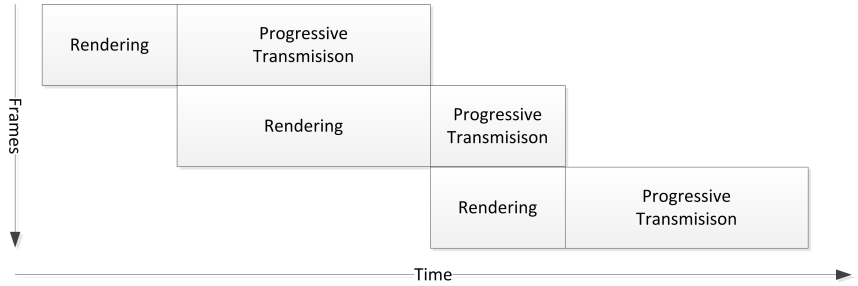


Figure 4.10: Pipelined execution: Rendering is performed concurrently to creating data structures, compression and transmission on the parallel renderer.

4.3.2 Progressive Image Streaming

In order to reduce the delay for a user waiting for a full resolution image, image streaming functionality are implemented for image and depth data to the framework. This feature is enabled by generating a data structure suitable to stream image data. Additionally the data structure should be able to give a quick preview of the whole image to locate interesting details for a deeper observation.

The progressive image streaming method is based on reordering remote images according to z-order curves. A z-order curve is a space-filling curve covering the whole domain of an image. The construction of a z-order curve is recursive, starting with the first level defined by one point. To fill-up space, each point of the already constructed curve is recursively substituted with four points aligned in a z-shape. Therefore, each recursion multiplies the number of points in the z-order curve by a factor of four.

Due to this construction of z-ordered curves, the defined point pattern always spans a quadratic region with a power-of-two edge length. Therefore, they can not directly be applied to non-power-of-two images, which are common for usual display resolutions. A common technique is to add padding points to fill up space to the next power-of-two resolution, adding a huge overhead in many cases. Furthermore, using one z-curve to cover an image requires increasing numbers of points to be transferred to update an image from one level to the next level.

To overcome these drawbacks, the image and depth buffers are divided into blocks of 8-by-8 pixels as depicted in figure 4.11. With this division, only few padding points need to be added to an image, especially, images of usual screen resolutions do not require padding at all.

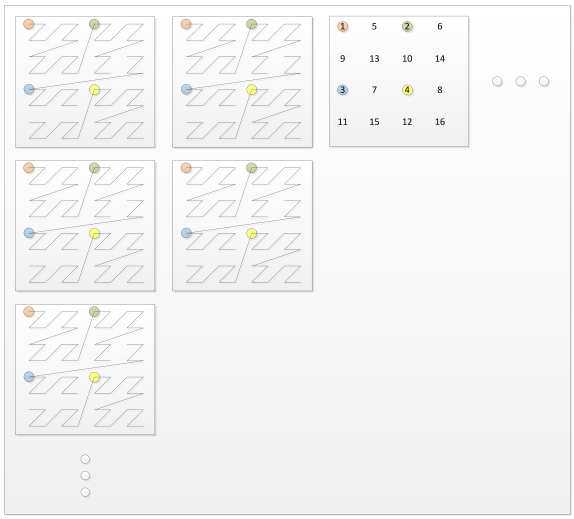


Figure 4.11: Progressive image data scheme. Remote images are divided into 8-by-8 blocks, each with a z-curve. According pixels are then combined with respect to their z-curve index.

In order to prepare for the progressive image streaming, according pixels of the z-order curves are combined in a packet, such that each of the first values are combined, each of the second ones and so forth. In this manner, we achieve a subdivision of the image space into packets of equal size.

Finally, progressive streaming of the remote image and depth buffers is achieved by transmitting each of the combined packets in the order of their z-order occurrence.

In order to save network bandwidth, each of the packets is compressed via standard zlib compression before transmitted to the front-ends. This streaming approach is demonstrated in figure 4.12. On the frontend side, remote rendered images are rendered by combining local and remote images by comparing the depth values for each pixel in an OpenGL fragment shader. Therefore, the depth and color textures containing the remote image are updated as soon as a packet is received and decompressed from the remote renderer.

To maintain interactive update rates of the remote rendered images for visual exploration, a new image with the current modelview transformation is requested from the remote renderer as soon as the first level of the progressive image was received.

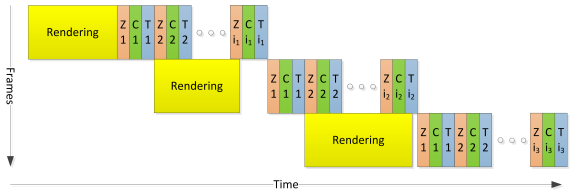


Figure 4.12: Progressive streaming pipeline. Each rendered image is subdivided into blocks according to their z-index, compressed and transmitted to the frontend as long as the next image is being rendered.

4.3.3 Results

The benchmarks that I will present in the following are similar to the benchmarks utilizing the powerwall system. They will demonstrate the benefits and enhancements of my streaming approach that connects tiled display wall systems with remote render solutions. Details about the RIG 250 data set can be found in section 4.2.2.1. The GPU cluster system is described in section 4.2.2.2.

4.3.3.1 Display-Wall Configuration

The available tiled display wall system consists of 4 by 3 displays, each of these screens having a resolution of 1920 by 1200 pixels. Every two screens of the entire display are connected to one computer for rendering. These machines are equipped

with one Intel Xeon E3-1245 quad-core processor with 16GB DDR3 main memory and one midrange NVIDIA Quadro 2000 graphics card with 1GB DDR5 graphics memory. The network connection to this cluster is 1Gbit Ethernet which is shared by all six machines.

Two different display setups were configured to run benchmarks with different numbers of render instances and tile resolutions. In the first configuration twelve frontend clients are launched on these six hosts, where each viewport has a resolution of 1920 by 1200 pixels. In the second configuration only six frontend instances are launched and the viewport resolution is doubled to 3840 by 1200 pixels. The total resolution of the display wall is 7680 by 3600 pixels.

4.3.3.2 Benchmarks

Figure 4.13 shows our visualization results on the tiled display wall at German Aerospace Center in Brunswick. The remotely rendered isosurface, colored by parallel process id, is composed with the gray context geometry on the frontend. In order to define a repeatable benchmark suite, likewise used for the benchmarks in section 4.2.2.3, this isosurface is rotated by 3.6 degrees per frame, whereas 50 frames are rendered in total. All benchmarks are repeated five times and the median of these five is kept.

When working with interactive visualizations, latency is a very important property. Therefore we measured the timings when first and last results are available after an user requested images for new positions. These timings are compared with the time required for rendering and compositing within IceT. This allows to determine the overhead of our implementation. The overhead is caused by the following tasks:

- Serialization and de-serialization of the image buffers and their meta data
- Re-ordering the original data structure to our progressive data structure
- Compression and decompression by the zlib library

Figure 4.14 shows the measured timings for the configuration when using six frontend instances connected to two GPU cluster nodes. The rendering is done with six GPUs

Figure 4.13: Display wall at German Aerospace Center in Brunswick running the used benchmark suite. Context geometry (gray) is rendered locally, isosurface is colored according to parallel process id.

on the remote renderer and a resolution of each 3840 by 1200 for each viewport. In average 520ms are required for the first image data being available at the frontend. Thereafter, first preview images can be rendered. Compared to the time required by IceT for compositing and rendering, an overhead of only 60ms in average is added by our implementation. Remote images are completely transferred after 1450ms in average.

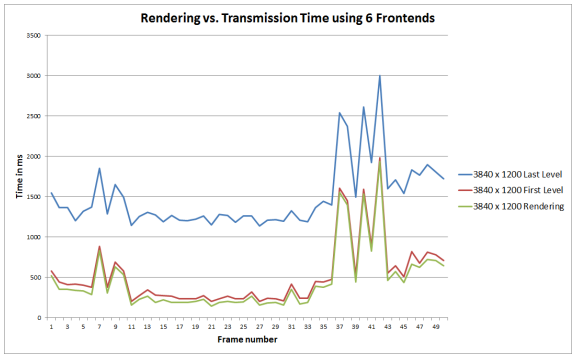


Figure 4.14: Measured timings when using 6 frontend instances connected to two GPU cluster nodes using 6 GPUs for rendering

In the second configuration, twelve frontend instances are connected to four GPU cluster nodes. The isosurface is now rendered with 12 GPUs and half of the viewport size compared to the first configuration. Figure 4.15 shows that the time the user is waiting for first results can be reduced to an average of 340ms. A complete update for the entire display wall now takes only 720ms which is half the time achieved with two GPU cluster nodes and six graphics cards.

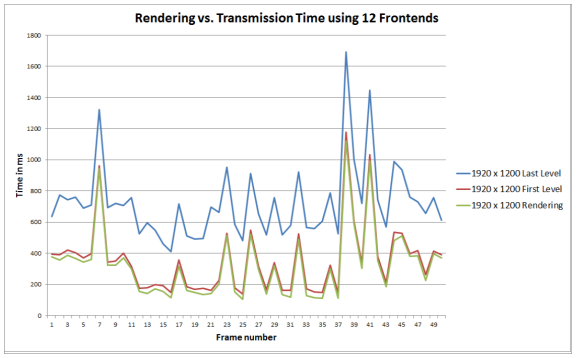


Figure 4.15: Measured timings when using 12 frontend instances connected to four GPU cluster nodes using 12 GPUs for rendering

The next figure 4.16 depicts the time required for the transmission of a complete remote image. When using six viewports with larger resolution and only 6 remote processes this time is about 930ms in average. Using the second configuration with twelve viewports the transmission time reduces to 380ms. This is caused by doubling the number of parallel instances. Therefore, each thread re-ordering the image to z-curve representations has to deal with half the amount of 8-by-8 blocks. Furthermore, the time required to compress network packets is growing more than linear.

Another important factor when comparing these timings is how many levels have been transferred until a new rendered image is available on the remote renderer for transmission. This means previous image frames are continuously updated while the camera is still moving. Figure 4.17 shows this scenario for the two different configurations. While in the 6 GPU configuration only a few images are updated to the full resolution in the 12 GPU configuration more images are reaching this last level due to better network utilization and lower overhead.

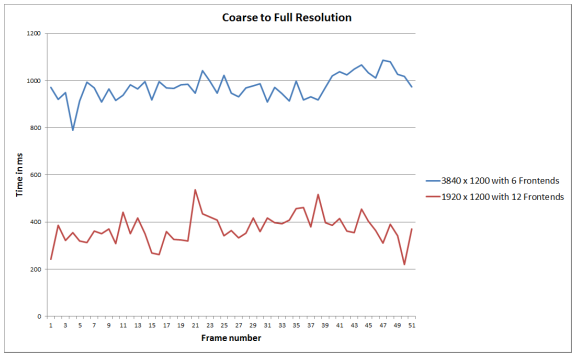


Figure 4.16: Time required to transfer the complete framebuffer.

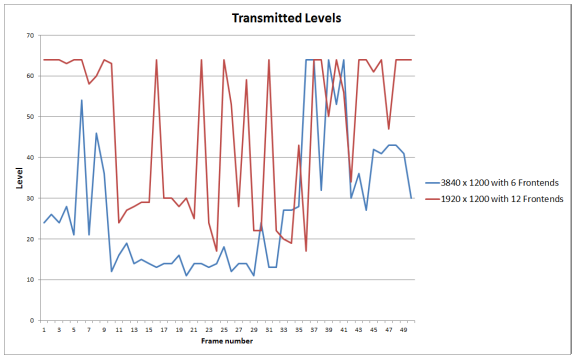


Figure 4.17: Number of transmitted levels of a frame until a new image has finished rendering on the remote renderer.

Because of the limited bandwidth between the GPU cluster and the display wall system, efficient compression algorithms are required. Here, a standard zlib compression was is applied for each data package. In Figure 4.18 the compressed frame buffer sizes for the entire display wall are proposed. In frames in which the isosurface is covering only some display tiles the compression rate is high and a compressed image takes about 18 MB. When the geometry is visible on all tiles the compression rate is decreasing resulting in image sizes about 23 MB. Without compression an update for one frame would require for the transmission of 210 megabytes.

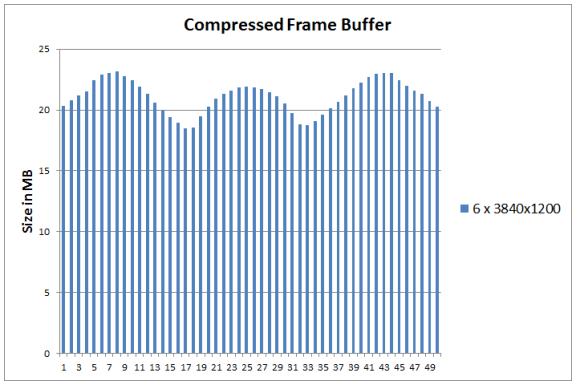


Figure 4.18: Size of compressed framebuffer in megabyte for the whole diplays wall at different frames

4.3.4 View-frustum culling

During the rendering pass, three-dimensional image adaptations, as described in section 4.2.1.3, lead to the fact, that possibly all remote viewports are visible on a single client's viewport. This means, that for each viewport all possible remote tiles have to be adjusted and rendered. In the case of large display wall systems, this leads to two important drawbacks. First, this would result in a strong rendering overhead, adjusting many pixels of the remote images which are possible not visible at all. Especially with high resolution displays, strong rendering load is burdened to the graphics processing units, and interactive frame rates are hard to achieve. And second, scalability to setups involving higher number of displays is lost.

To reduce requirements on the available rendering performance, I implemented a view-frustum culling optimization. Therefore, each of the remote depth images is divided into smaller parts, each part 128 by 128 pixels. Thus, the complete remote image covering all viewports consists of 1800 image parts. For each of the image parts, minimum and maximum z-depth values are determined, and a bounding box of all depth pixels inside a remote image tile can be defined. On the front end, a remote image tile only needs to be rendered if the adapted bounding box is visible on the current display viewport.

Figure 4.19 shows the number of rendered tiles per front end client in a session with a head-tracked user, revealing a significant reduction in rendering overhead.

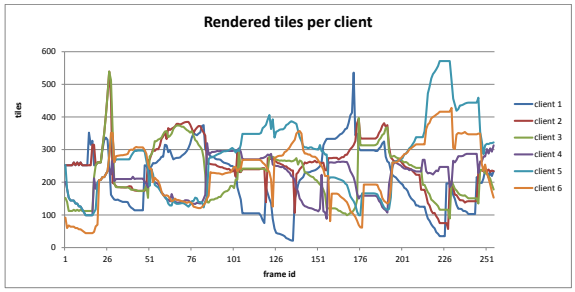


Figure 4.19: View-frustum-culling of adapted remote rendered image. The remote image is divided into smaller parts (1800 in total), each with its bounding box. These smaller tiles are rendered on a client, only if the adapted bounding box is covering its display viewport.

4.3.5 Discussion

In this section, I presented my hybrid rendering solution to enable for interactive exploration of online monitored simulations using tiled display walls was presented. Therefore, a context geometry is rendered locally, while complex feature are rendered remotely. In order to transfer the remote image, a multi-resolution, progressive image streaming approach based on z-order curves is used.

We showed that our approach is capable to support interactive navigation through the dataset at high frame rates all the time and remote rendered images were available on the display wall with low latencies. Our z-curve multi-resolution scheme enabled for an explorative overview. During our benchmarks, the remote render performance changed according to the current image. Our solution was capable to automatically adapt to these variations and during longer rendering times more details are transferred to and displayed on our tiled display wall frontend.

Our findings show that hybrid rendering and multi-resolution progressive image streaming is a viable solution for using high-resolution tiled display walls. This approach is promising to be useful for online monitoring of future HPC simulations, if intermediate post-processing data is available to a parallel remote renderer.

4.4 Hybrid Rendering of Time-Dependent Simulation Data

Simulations of dynamic systems lead to data sets with high spatial and, especially, high temporal resolution. Standalone interactive exploration workstations often cannot handle the amount of information that is generated in those simulations. They either suffer from exceeding local memory capacities which are insufficient to keep all necessary data set parts or have insufficient disc I/O performance to reload data quickly.

My main contribution which I introduce in this section is the presentation of a hybrid rendering approach that enables for handling large time-dependent data sets. In this section I include loading and interactive feature extraction to the remote rendering process, while I still guarantee interactive frame rates and interactive navigation. The will successfully demonstrate the results with simulation data sets of the SHEFEX I flight experiment by the German Aerospace Center which investigates the behavior of new shapes of space crafts during the re-entry phase into earth's atmosphere.

In order to meet the requirements of analyzing high spatial resolution data sets, I extended my remote post-processing and rendering system already presented in section 4.2.1. As context geometry, I render the motion dynamics of the flight body on the local visualization workstation equipped with virtual reality techniques enabling interactive explorations. A pixel image of the remotely rendered isosurfaces is composed with the locally rendered flight body geometry, as soon as available. Because of network latency and the ongoing time-dependent visualization, the time stamp of both current images is diverse resulting in mismatched geometries depicted in the exaggerated view in Figure 4.20. Furthermore, splitting rendering workload into parts for local and remote resources enables my framework for introducing additional post-processing steps without loosing interactivity. Here, I compute isosurfaces used to render shock waves on the fly which allows for interactive iso-value selection. The fact that my approach reduces the temporal dependencies of remote and local rendering can be exploited to parallelize the rendering of different time-steps. This means, that my approach is capable of rendering images for multiple

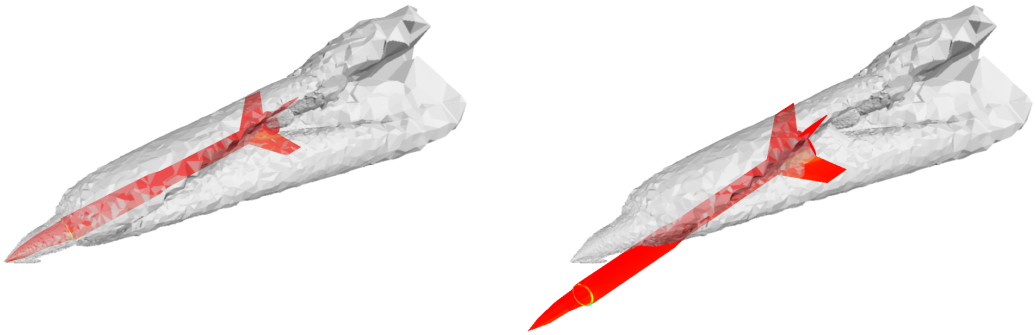


Figure 4.20: When combining locally and remotely rendered images with different update rates, the time stamp of both is not guaranteed to be equal. Instead of the correct composed images (left), locally and remotely rendered images can diverge (right).

time-steps on the GPU cluster nodes in parallel which directly results in higher remote update rates.

In the following, I will present the SHEFEX I application example followed by the description of necessary pipeline modifications to my former described framework. Thereafter, I will present and discuss results that demonstrate the successful interactive navigation and visualization using my hybrid rendering approach.

4.4.1 Application Example – SHEFEX I

Investigating the behavior of spacecraft vehicles during the re-entry phase into the atmosphere of planets is one of the examples where simulations with a very high dynamic in time and space are performed. Currently, the German Aerospace Center (DLR) is working on the development of new shapes for re-entry spacecrafts. In the program SHEFEX (Sharp Edge Flight Experiment), the first experiment was already launched in 2005 [THJ+06] (cf. Figure 4.21). In order to get a better understanding of the influence of the shape on its parabolic flight path and during the re-entry into the earth's atmosphere, not only windtunnel experiments but also numerical simulations (computational fluid dynamics, CFD) [BL10] were carried out. It turned out that the flight body showed a complex twisting motion during the re-entry phase

which could only be adequately reproduced with a CFD simulation coupled to a flight dynamics solver at high temporal resolution [BCE11]. For the interactive analysis of the simulation results in a virtual environment, the flight body motion has to be visualized with a high frame rate.

Figure 4.21: Re-Entry of SHEFEX I (illustrated by DLR)

One of the important flow features is the shock cone appearing around space vehicles flying with hyper-sonic speed. Isosurfaces defined by iso-values with respect to the simulation time are used to depict the shock wave. However, compared to the rigid body motion the shock provides less changing information over time and therefore does not require the same high update rates for a fluent perception as the spacecraft motion does.

4.4.2 Modified Framework Architecture

This section presents the modification to my prior presented hybrid rendering framework, as described in section 4.2.1, in order to provide rendering of the dynamic motions of the SHEFEX I flight.

The rendering system itself (cf. figure 4.22) is a distributed system that consists of a front-end workstation or virtual reality system and a remote render back-end equipped with high performance GPUs. The front-end requests and stores images from the back-end. The back-end responses to these requests with remote rendered color images including attached depth buffers.

In order to meet the interactivity requirements, my hybrid rendering approach that splits the visualization workload and combines local and remote images. Therefore,

context geometry is rendered on the front-end and the expensive extraction and rendering of the iso-surfaces is done on the back-end.

The front-end updates the context geometry in each frame. Rendering the context geometry is done in two passes. The first pass renders the flight path in order to provided spatial path context information. This path is illustrated with a surface spanned by the flight bodies main y-axis. The second pass renders the moving SHEFEX I surface mesh. Local rendering performance is almost constant and supports high frame rates.

The back-end receives rendering requests from the front-end, loads the required data, extracts and renders an iso-surface, and sends a compressed result image back to the front-end. The update rates I could achieve with one remote rendering back-end is slower than the local frame rate. Using multiple remote render instances, the update rate can be increased, however, with the four available GPU nodes the remote frame rate is still below the local frame rate. Since network transmission is inherent in rendering remote images, they are available at the front-end with delay. Therefore, in order to have remote images available for the current local time-step, the front-end requests them for future time steps.

Finally, local and remote images are combined. Therefore, the local rendering chooses the stored remote image with the nearest time stamp compared to the current visualization time. While remote images update less frequently and camera position continuously change in virtual environments, visual information in local and remote images usually does not fit. Therefore, remote images are adjusted to the current view.

In summary, the main tasks for my remote renderer are

- loading of time-dependent data,
- extraction and rendering of iso-surfaces,
- compression and transmission of remote images to the local rendering workstation,

while the local renderer needs to deal with

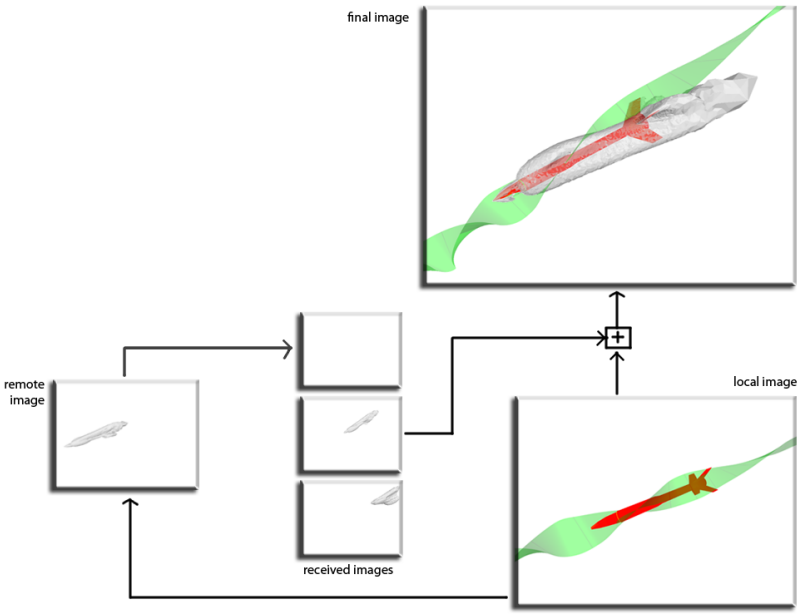


Figure 4.22: The hybrid rendering approach updates remote rendered images for future time steps and combines one of them with the local image at high frame rates.

- organizing and requesting remote images,
- rendering local geometry and
- re-adjusting remote images according to the current view.

4.4.2.1 Pipeline

In order to generate requested images for the local virtual environment, my parallel remote renderer, which is based on the IceT framework, is extended by pipelining. This pipeline, as depicted in figure 4.23, is paralleled by multi-threading.

Disc I/O is a main bottleneck for this kind of time-dependent visualizations. Using a dedicated pipeline stage to load data results in pre-fetching capabilities. If new data is available it is passed to the *algorithm/rendering* stage. This stage executes a user algorithm, here iso-surface extraction and rendering. The rendered output is composited by IceT, storing it on the master node. The last stage, *image com-*

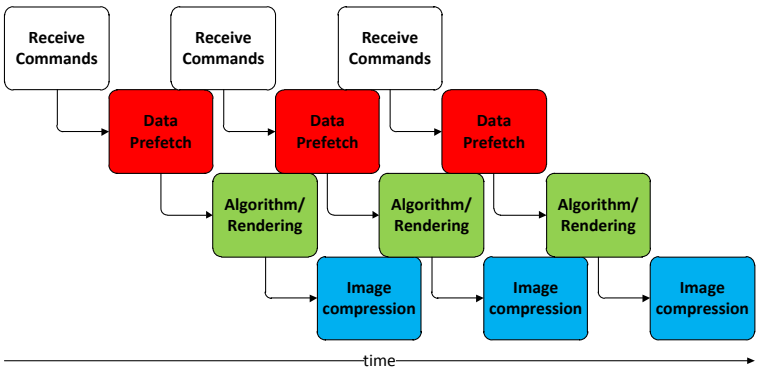


Figure 4.23: The pipelined execution in our remote renderer supports overlapping execution of data pre-loading, rendering, and image compression.

pression, compresses image and depth buffers using the zlib library. The lossless compressed data is finally sent to the local front-end.

4.4.2.2 Isosurface Extraction

An isosurface is the three-dimensional surface representing points of constant value. Isosurface visualizations are utilized in order to represent individual shock waves during the SHEFEX I supersonic flight. Isosurface extraction is a topic deeply investigated in the past. Well-known algorithms to extract isosurface representations from scalar fields are the Marching Cubes algorithm [LC87] as well as the Marching Tetrahedron algorithm [Bou97]. Different hardware-accelerated versions utilizing GPUs have been developed, including versions using nVidia's Compute Unified Device Architecture (CUDA). While many evolved versions exist to handle large tetrahedral meshes [KSE04, Pas04] or arbitrary meshes [JC06], the CUDA toolkit also comes with a trivial marching cubes implementation. As presented in the benchmark section 4.4.3.1, isosurface extraction required only a small fraction of time for the SHEFEX I data set. Therefore, it is sufficient that we make use of a trivial adaption of nVidia's version.

In general, time-dependent iso-surface extractions for unstructured grids involve loading scalar data as well as point and cell information for each time step. In the

SHEFEX I simulation the vertex positions are moving, however, the mesh does not deform during simulation time. Thus, we can exploit this by loading the mesh only once, saving a lot of I/O time. The vertex positions for different time steps are determined by applying an additional model matrix M_{rigid} that describes the rigid body motion. This matrix M_{rigid} is passed to the back-end in the render command in combination with time-step and iso-value information. By applying M_{rigid}, the model view projection matrix MVP changes to

$$MVP'_{remote} = MVP_{remote} \cdot M_{rigid}.$$

After uploading scalar data for a particular time step to CUDA memory, rendering the isosurface becomes a five pass algorithms. In the first pass, a CUDA kernel determines the active cells, containing the current iso-value, and the possible triangle counts for each cell.

In a second pass, the number of active cells and triangles are determined as well as the active cell indices are collected in a consecutive array. This is achieved by using a standard prefix-sum-algorithm supporting by the thrust-library, which is part of CUDA.

The following third pass calculates and stores the triangle and normal information into vertex buffer objects using another CUDA kernel.

While the vertex buffer objects are then bind to OpenGL and rendered in the fourth pass, the resulting frame buffer gets finally grabbed and composited by the remote rendering framework.

4.4.2.3 Image Adjustment

Similar to the last sections, the camera positions between locally rendered images might change and the image features displayed on both images do not match anymore. Furthermore, the geometry of the flight body is moving due to time-dependent animations. Therefore, in contrast to the previous application examples, remote and local renderings diverge even if the viewer position is fixed. In order to re-adjust local and remote images, we apply the image adjustment techniques described in

section 4.2.1.3. In summary, pixels p of remote images are transformed to pixel positions p' according to the current viewing position by

$$p' = T_{local}(T^{-1}_{remote}(p)).$$

The application of the image adjustments to the SHEFEX I simulation is presented in figure 4.24. Figure 4.24 (a) depicts a situation where both images do match and the camera is rotating around the scene in figure 4.24 (b).

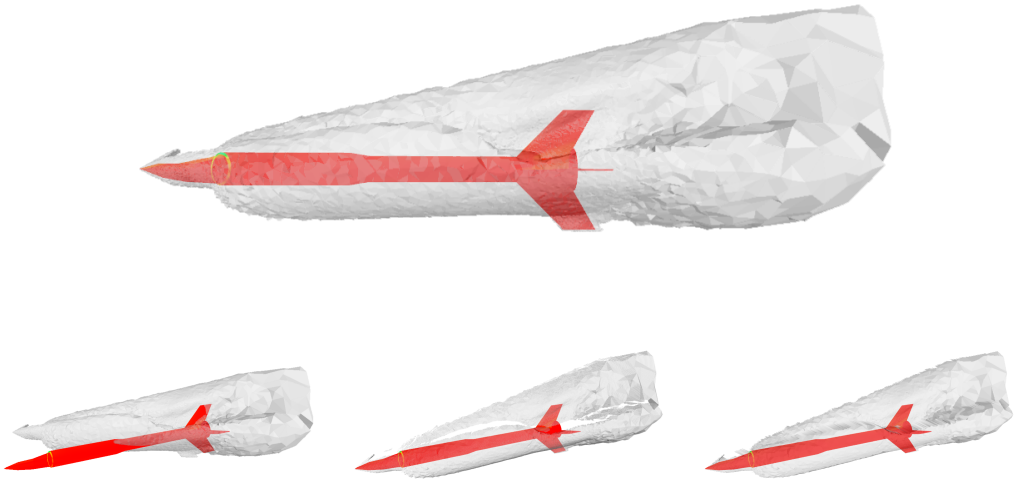


Figure 4.24: Exaggerated example of locally rendered geometry and a remote rendered isosurface. While matching in the original time step (above), the view is rotated in the lower images. By using simple texture combining (lower left), the remote image is not adapted at all. Point based rendering (lower middle) and mesh based rendering (lower right) adapts the remote image. While point based rendering suffers from wholes in the result, mesh based rendering connects possibly not connected surface components.

Figure 4.24 (c) and (d) show the examples of two image adjustment strategies. First, figure 4.24 (c) depicts an example of in which point rendering is used. Second, figure 4.24 (d) shows a rendering in which neighboring pixels are used to render a quad mesh instead of isolated points.

Both approaches are capable to re-adjust the remote rendered images to the current camera position. However, both approaches have visual artifacts. On the one hand, the point rendered iso-surface suffers from missing pixels in the result, especially at zoomed views, these wholes are filled with the quad rendered approach. On the other hand, the quad rendering might connect iso-surface pixels, that were not connected in the original image.

4.4.2.4 Requesting future Time-Steps

Remote images requested from the back-end return to the front-end with latency. Therefore, in order to have remote images available at time, the front-end requests them for future time steps. In the following, I present the heuristic that I implemented to determine the time-step of which the remote renderer should generate an image.

Let t_{local} be the time at which a remote request is finished and a next request will be prepared. The front-end is determining the averaged delay Δt of the last finished remote requests. In the case of using one remote renderer, the time t_{remote} in the generated new request will have an offset of twice this delay,

$$t_{remote} = t_{local} + 2\Delta t.$$

The update rates measured in the benchmark, cf. section 4.4.3.2, reveal a performance of about 5 frames per second in average. Multiple render instances, if available, are used in order to increase the update rate, each of them rendering different time steps.

For the implementation, the front-end is connecting to each of the n master nodes of each remote renderer. Furthermore, the generation of image requests to the back-ends is changed slightly. To the time offset for each request an fraction according to the renderer id, $id = 0, \ldots, n-1$, is added,

$$t_{remote} = t_{local} + 2\Delta t + \frac{id}{n}\Delta t.$$

4.4.3 Results

To evaluate the performance of the time-dependent hybrid rendering approach, we employed the following benchmark utilizing a front-end workstation and a four-node GPU cluster.

The local rendering was performed on a workstation with an Intel Xeon E5520 2.27 GHz quad-core processor, equipped with 24GB of RAM. The remote GPU cluster, the same as used in section 4.2.2.3, consisted of four nodes, each with a dual Intel Xeon X5670 2.93 GHz hexa-core processor, 48GB of RAM and three NVIDIA Quadro 6000 6GB graphics cards. A 1GBit ethernet network is connecting all involved compute nodes. The requested RGBA images had an image size of 1024x768 with an attached 32-bit depth buffer.

The SHEFEX I simulation data used to perform this benchmark consists of 2614 time steps with non-uniform sampling time. These time steps represent the complex twisted motion during the re-entry phase taking place from second 419 to second 439 after launch. The spatial domain around the flight body was approximated with a mesh consisting of 660 666 vertices and 3 343 026 cells. While the mesh can be re-used for each time step, the storage system has to update 15.8 MB per rendered time-step. In total 41.3 GByte have to be loaded and processed.

4.4.3.1 Scheduling and Latency

Figure 4.25 presents in detail the workflow of a single pipelined remote rendering solution for some consecutive render commands during the visualization. The overlap of consecutive tasks are clearly visible. In this benchmark, disc I/O is the most dominant bottleneck. This is visible by the fact, that loading data from disc is performed continuously. The execution of isosurface extraction and rendering (green bars), image compositing (yellow bars), and image compression(light blue bars) is done completely in parallel to the disc-I/O. Therefore, a rendering as fast as data could be read from disc is achieved.

Nevertheless, for each remote image, the latency between requesting remote images and their availability to the local front-end is determined by the total amount of

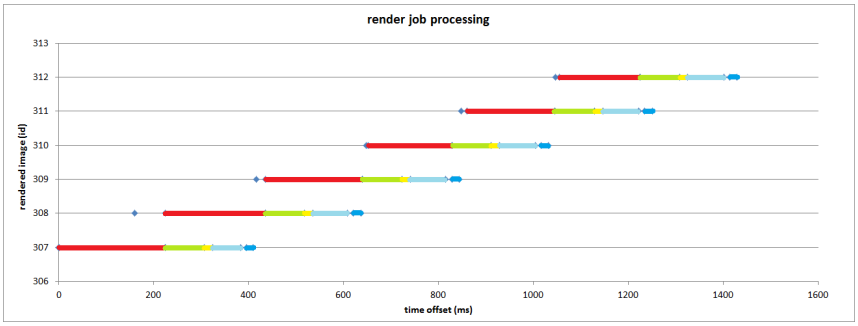


Figure 4.25: Scheduling of overlapping render jobs for a subset of remote request. Overlap is visible between time of request (blue dot), disc I/O (red), isosurface extraction and rendering (green), image composition (yellow), compression on back-end (light blue), and decompression on front-end (dark blue).

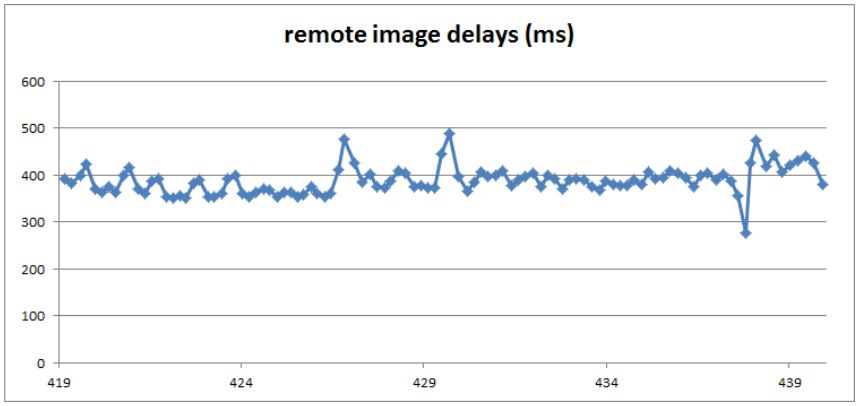


Figure 4.26: Delay between remote requests and availability during interesting animation times. A nearly constant delay of about 400ms was measured.

the disc I/O operations, rendering, image compositing, and compression as well as the decompression on the local system. Figure 4.26 shows the delay of the remote rendering system during an animation. A relatively constant delay of about 400ms was measured.

4.4.3.2 Render Performance

In this section, I make use of hybrid rendering in order to decouple the rendering of smaller-sized context geometry at high frame rates from heavy post-processing workload of time-dependent simulation data sets. Despite the fact that this division

supports local render performance at interactive frame rates, post-processing and rendering can be paralleled over different time steps. The behavior of the presented approach utilizing a varying number of GPU cluster nodes is depicted in figure 4.27.

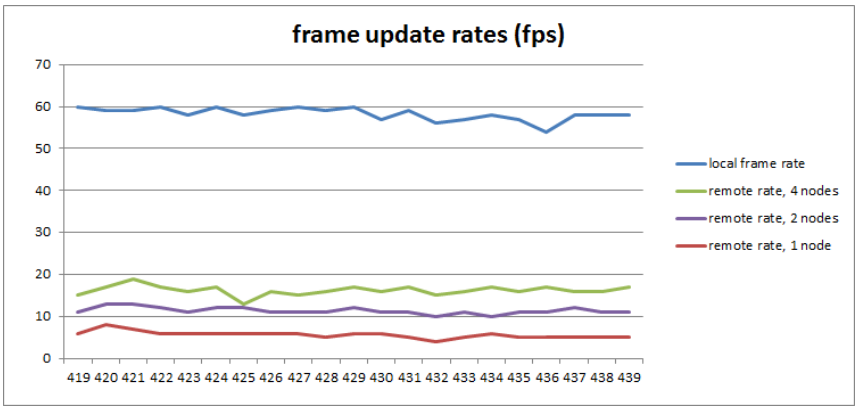


Figure 4.27: Update rates of local and remote renderings. While a local update rate of about 60 fps supports interactive animation of dynamic motions, remote rendered images are updated less frequently.

The local frame rate is 55 fps at minimum, sufficient to recognize all details of the flight body motion. Due to the fact that the isosurface is extracted and rendered concurrently, their update rate is independent of the local rendering. An average of 5.6 fps was achieved utilizing one GPU cluster node and 11.3 fps with two GPU cluster nodes. When all four GPU cluster nodes were used the remote update rate had an average of 16.2 fps.

4.4.4 Discussion

In this section, I presented the application of my hybrid rendering approach in order to interactively explore simulation data sets with high spatial and temporal resolution. My approach divides the rendering workload into rendering of geometric context components and rendering complex flow features such as time-dependent iso-surface visualizations. While geometric context components can be rendered and animated locally with high frame rates complex features are decoupled and rendered remotely. My presented system handles these divergences by re-adjusting remote rendered images with image-based rendering adaptations.

The application to dynamic motions, such as the presented re-entry flight body motion, emphasizes the importance and success of my technique. Due to the resulting interactive frame rates, I am able to provide clearly perceivable dynamic motion patterns. My hybrid rendering approach combines this animation at high frame rates with valuable context information such as complex shock cone iso-surfaces.

However, even if rendering remote images is decoupled and is not affecting local rendering, remote extraction and rendering can only support limited frame rates depending on the available hardware as well as the data set to inspect. Here, my hybrid rendering approach introduces a huge advantage which is clearly presented in the result section. Due to the division of the rendering workload it is possible to introduce an additional level of parallelism in the time domain. Therefore, multiple render instances, if available, can concurrently generate images for different time-steps resulting in much higher remote frame rates.

Future work in the field of hybrid rendering time-dependent simulation data will focus on the extension with geometry streaming. When the visualization time is stopped by the viewer, instead of continuously streaming new remote images, progressive streaming of the current iso-surface would enable a rendering without artifacts and at higher update rates of the formerly remotely rendered images.

4.5 Conclusions

Chapter 4 focused on the interactive navigation through large-scale data sets in virtual environments. When exploring large-scale simulations in virtual environments, the vast amount of data produced by nowadays and future high-performance computing resources, the requirements for interactivity burdens major challenges: (1) the size of data sets or extracted features exceeds the resources of local render hardware, (2) the increasing number of pixels enables to inspect finer simulation details on the one hand, on the other hand bandwidth can be a strong limiting factor for distributed systems, (3) time-dependent simulations amplify these requirements and add further emphases on data handling and interactive rendering. This chapter inspected hybrid rendering as a solution to all these issues.

Section 4.2 introduced my hybrid rendering approach for navigating through large-scale simulation data sets in virtual environments. In such environments, interactivity puts strong requirements on frame rates and therefore limits the amount of rendering primitives. My presented approach deals with this limitation, which is easily exceeded by large-scale simulation data sets. I enable the utilization of parallel remote render resources for heavy rendering workload. In addition, I provide a context geometry rendered at high frame rates on the front end, which supports interactivity. The downside of this approach is the high latency introduced by remote render solutions which is addressed by re-adjusting remote images to the current local view. I could clearly point out that this approach leads to successful navigation through large-scale data sets. The technique was tested on a three-pipe powerwall system and could achieve high local frame rates sufficient for interactive navigation.

Section 4.3 focused on the problem arising with increasing display resolutions. While finer details can be presented on high-resolution displays high network bandwidth is required in order to transmit remote rendered images. I showed that using progressive image streaming techniques are able to increase interactivity in this circumstance. My system implemented a progressive image streaming based on z-order curves and is tested on a twelve monitor tiled-display wall. This image streaming method is especially beneficial if remote render times differ a lot, since it automatically adapts to different rendering latency. Here, I could achieve higher update rates on tiled display wall systems which improve the ability to interactively navigate through the data set and find interesting regions.

Finally, in section 4.4 I tackled the requirements of time-dependent data sets. Which multiple time steps to consider, data set sizes often multiply by a large factor. Therefore, not only video memory but also random access memory of local rendering hardware is easily exceeded. The solution I presented here not only removed these hardware limitations from the front-end machines, moreover, the combination of local and remote images enabled for rendering multiple remote images in parallel. The main drawback of this approach might be its limited applicability to certain simulations. While local rendered parts can be animated with high frame rates, the update rates of remote rendered features depend very much on data set sizes and available remote rendering hardware. Even if remote rendered images are adapted

to the local viewer, it must be considered if remote update rates are sufficient to grasp their time-dependent information.

Chapter 5

Harmonic Analysis in Computational Fluid Dynamics

Among a multitude of techniques for the visualization of scalar, vector, and tensor fields, feature analysis methods play a crucial role in enabling the visualization of large datasets. Here, application-oriented feature definitions are matched against a dataset to highlight interesting regions. Especially for highly complex fields arising from modern computer simulations, visualization efforts can be reduced or made feasible by depicting significant structural components and their interactions, allowing for an abstracted view. Existing methods are typically based on topological structures or application specific feature definitions. While the former leverage a deep mathematical framework to generate a topological skeleton of a field and are uniformly applicable to general fields, the latter requires intimate knowledge of the application domain. Here, I apply concepts of harmonic analysis towards this goal. In general, applications of harmonic analysis (often also called signal processing) can be manifold. Range from simple and generic processing such as smoothing over a variety of computer vision algorithms, it can also be applied to the extraction, detection and classification of features.

In this chapter, I provide an introduction to harmonic analysis of vector fields and describe an application to feature-based visualization. As is required for a basic understanding of the concepts I discuss, I attempt to summarize harmonic analysis

techniques for discrete field representation. Harmonic analysis as a general concept is neither specific to application domains, nor is it restricted to specific field types (scalar, vector, tensor) or domain geometry. I illustrate a global analysis approach for generic fields and discuss its computational implications, which leads to define local approaches that are much more feasible computationally.

To achieve the goal of feature-based visualization, I define a feature space over small neighborhoods of a given discretized field's domain that transforms it – using harmonic analysis – into a low dimensional feature vector. This transformation is achieved by formulating a discrete Laplacian over the discretization of the neighborhood and computing eigenvalue decomposition. This yields a basis of eigenfunctions over the neighborhood. The coefficients of the field representation in this basis form the feature vector. The latter then provides a means to define, locate and compare features in an empirical fashion. My method is closely related to Fourier analysis that is used extensively e.g. in image processing and computer vision applications.

5.1 Related Work

Harmonic analysis is a concept with a rich and well-developed mathematical background, and has many applications. For example, in disciplines related to visualization, applications to geometry processing and mesh filtering have been discussed in depth, such as surface quadrangulation [DtBG06, TACSD06] or the design of tangent vector fields over surfaces [FS07]. Vallet and Lévy [VL08a] provide an overview of recent results. In the following, I concentrate on previous work that is immediately relevant to the practical presentation in this paper.

Eigenanalysis techniques are often applied in the context of Fourier transformations, convolution or pattern matching. Since classical Fourier techniques are not applicable to vector or tensor fields in a direct and meaningful manner, [SHM+07, ES03, ES05a] employ complex invariant moments or Clifford algebra in order to define a suitable setting. A different approach is based on a discrete formulation of the Laplacian operator, which is the central concept behind harmonic analysis, such that its eigenfunctions can be directly formulated for vector fields.

However, discrete Laplacian formulations, such as the Discrete Exterior Calculus (DEC) approach introduced by Hirani [Hir03], are not yet well understood regarding their application to vector-valued eigenvalue problems. Therefore, I compare numerical results with a classical Finite Element formulation of the Laplacian, for which many good textbooks are available like [HDSB08] as well as for eigenvalue problems itself [Hea02] and the underlying functional calculus [Kre78].

The implementations underlying my experiments (cf. Section 5.4) are based on the use of the ARPACK [LSY98,Sor96] and SUPERLU [DGL97] packages that facilitate eigenanalysis and decomposition of large matrices.

5.2 Harmonic Analysis

To provide a base understanding of my methods, I will briefly touch on and illustrate a number of fundamental concepts. Beginning with the Fourier decomposition as a direct example, I will discuss the spectral theorem known from functional calculus. Then, a short discussion about arbitrary domains and field types is followed by describing low pass filtering as a global approach. I will limit myself to a high-level, phenomenological overview and refer the interested reader to [Kat04, DP11] for an in-depth treatment of the topic.

5.2.1 Fourier Decomposition

The well-known technique of Fourier analysis is an example of harmonic analysis techniques, and I will discuss it here briefly to motivate the use of harmonic analysis on generic field types and geometries as a tool with manifold capabilities.

A periodic signal can be decomposed into a combination of sinusoidal functions with varying frequency and amplitude. Mathematically, sine and cosine functions are used as basis for the space of periodic functions, and the original signal is transformed

to this basis. Specifically, a periodic function f with periodic length $T > 0$ can be described as a sum of sine and cosine functions

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k \cdot \cos(k\omega t) + b_k \cdot \sin(k\omega t)),$$

where the coefficients a_k and b_k are given by the projection of the function f onto the Fourier basis functions as

$$a_k = \frac{2}{T}\int_c^{c+T} f(t) \cdot \cos(k\omega t)\,\mathrm{d}t$$

and

$$b_k = \frac{2}{T}\int_c^{c+T} f(t) \cdot \sin(k\omega t)\,\mathrm{d}t.$$

Furthermore, these sinusoidal functions can also be interpreted as the real and imaginary parts of the complex valued functions e^{-ikt}. These are specifically the eigenfunctions to the second derivative operator, i.e.

$$-\frac{\partial^2 e^{-ikt}}{\partial^2 t} = -i^2k^2e^{-ikt} = \lambda e^{-ikt}.$$

In this simple case, the second derivative operator is the specific form of the Laplacian operator on the one-dimensional periodic domain $[0, T]$.

In summary, the Fourier decomposition is a Laplacian eigenvalue decomposition, where a periodic function is represented as a linear combination in a basis of eigenfunctions of the Laplacian operator.

5.2.2 Spectral Theorem

The mathematical background I am using has its origins in the spectral theorem of functional calculus. The theorem provides a strong and useful relationship between an operator $\mathcal{T} : V \to V$, e.g. $V = \mathbb{R}^n$, and its eigenfunctions with

$$V = \ker \mathcal{T} \oplus \overline{\mathrm{lin}}\{e_1, e_2, \ldots\}.$$

In a nutshell, for a compact operator $\mathcal{T}$ on a function space (such as the Laplacian), the spectral theorem states that the function space can be decomposed into a direct sum of the operator's kernel (which maps functions to zero) and the linear space spanned by its eigenfunctions. The non-zero functions e_i and non-zero values λ_i which fulfill

$$\mathcal{T} e_i = \lambda_i e_i,$$

are called the *eigenfunctions* e_i to the corresponding *eigenvalues* λ_i of the operator $\mathcal{T}$. The set of λ_i is called the *spectrum* $\sigma(\mathcal{T})$ of the operator.

In the following, the Laplacian Δ will be the central operator I am concerned with. Since the Laplacian is a symmetric operator, its eigenfunctions are orthogonal, and the projection of a function onto the Laplacian's eigenfunctions is the simple inner product (typically in a L_2-sense) of two continuous functions f and g,

$$< f, g > := \int f(x) \cdot g(x) \, \mathrm{d}x.$$

Consequently, the projection of the function f on the eigenfunctions e_i results in the basis coefficient a_i and has the continuous version

$$a_i = < f, e_i > := \int f(x) \cdot e_i(x) \, \mathrm{d}x.$$

5.2.3 Discrete Setting

In practical applications, one has to consider a discrete setting, where a field under consideration is described over a computational grid. Here, the Laplacian Δ is represented by a matrix A that describes its action on a discrete representation of a given field. Its eigenfunctions are the eigenvectors e of A and the spectrum is given by

$$\sigma(A) = \{\lambda \in \mathbb{C} \,|\, \exists e \neq 0 : Ae = \lambda e\}.$$

The projection in the discrete case simplifies to the sum

$$a_i = x^T \cdot w \cdot e_i = \sum_{j=1}^{n} x_j \, w_j \, (e_i)_j,$$

where x^T is the function vector to be projected, e_i the i-th eigenvector of A and w is an area weighting function given by the size of the area around each vertex j.

5.2.4 Arbitrary Domain and Field Type

As motivated in Section 5.2.1, one aspect of harmonic analysis is the study of Laplacian eigenfunctions and eigenvalues. For the classical Fourier decomposition the functions are scalar-valued and periodic. Thus, they can be interpreted as scalar functions on a simple manifold: the unit circle S^1. However, the definition of the Laplacian is more general and can be stated in other circumstances. Especially, it can be defined for scalar, vector or tensor fields, and over different domain geometries including arbitrarily-shaped manifolds. Figure 5.1 shows an example of formulating the Laplacian for vector fields on an irregular shaped region, and the corresponding first eigenfunctions (in this case a vector field) are shown.

5.2.5 Global Analysis

In similarity to the Fourier decomposition, harmonic analysis can be applied to an input field's global domain. I term this a *global approach* in the following.

As an application example, I consider a turbulent vector field dataset arising from a CFD simulation and described on an unstructured mesh with about 50,000 vertices. The resolution of the mesh is slightly adaptive with vertex distance decreasing in the middle of the domain. As in this example, vector fields arising from e.g. computational fluid dynamics solvers, are often given on arbitrary shaped geometries discretized on unstructured grids. As outlined above, this yields the necessity of discretizing the Laplacian on the underlying discrete function space and solving for eigenvalues and eigenfunctions of the resulting matrix representation explicitly. I discuss details of the discretization process in Sections 5.4.1 and 5.4.2, and the eigenvalue computation is discussed in Section 5.4.4.

After the eigenvalues of the matrix resulting from the discretization of the Laplacian over the domain of consideration are computed, any spectral method operating on the eigenfunctions can be applied and typically modifies the representation of the

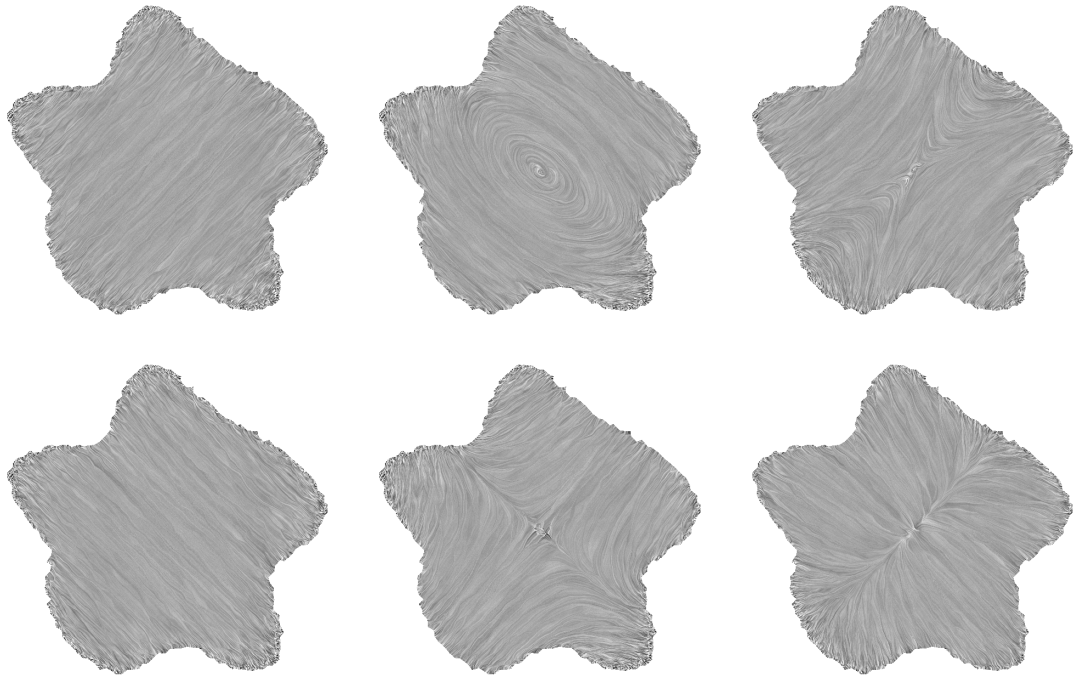


Figure 5.1: Example of vector-valued Laplacian eigenfunctions, which are vector fields themselves, illustrated by Line Integral Convolution. Eigenfunctions of the vector-valued Laplacian are shown on an irregular-shaped region with cells on an unstructured grid. The eigenvalue multiplicity for the shown eigenvalues is two, therefore two corresponding eigenvector fields (top row vs. bottom row) are orthogonal.

field in this basis. The resulting linear combination of eigenfunctions can then be evaluated on the original basis (e.g. per vertex) to allow interpretation of the result in the original context. Using this approach, Figure 5.2 illustrates the effect of applying a low pass filter to the vector field that dampens the high frequency vector field oscillations by reducing the coefficients of eigenfunctions proportional to the corresponding eigenvalue magnitude.

Similarly, high-pass or band-bass filters can be applied to enhance different components of the vector fields. For example, increasing higher frequencies can amplify the vorticity to allow easier exploration of small scale structures.

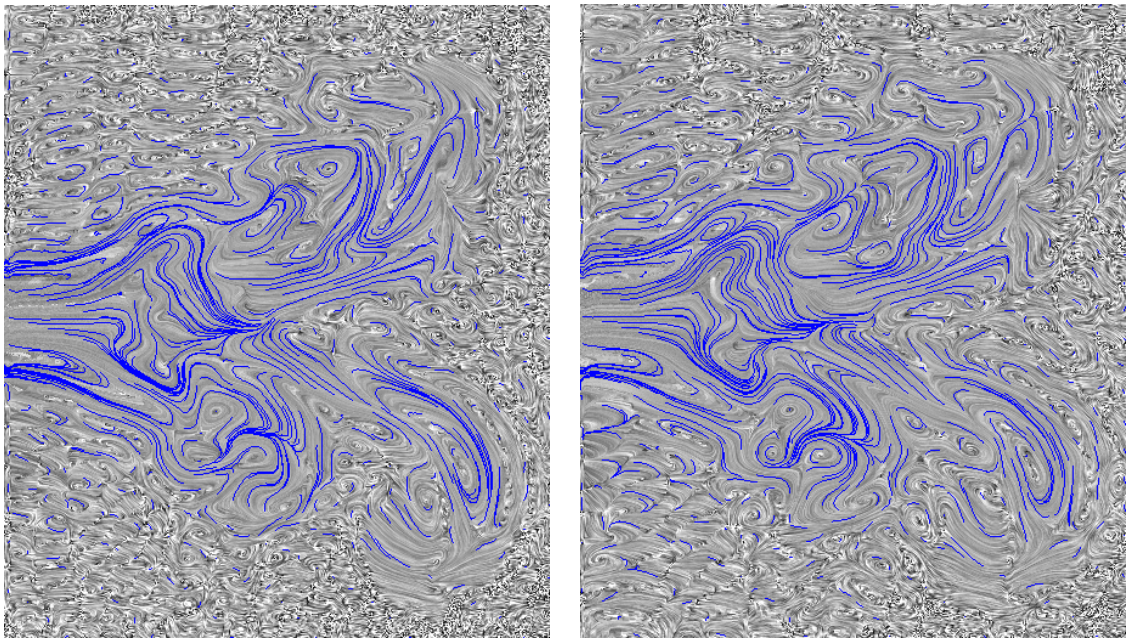


Figure 5.2: The spectrum of the left vector field is dampened by removing high eigenvalue terms resulting in the right vector field. Some streamlines are drawn in the vector field to illustrate small turbulent structures.

5.3 Local Harmonic Analysis

The dominating problem of global approaches is the large and significant computational cost arising from the computation of the Laplacian eigenvector basis. Since the storage space and calculation complexity grows quadratically in the number of vertices, global approaches are quickly limited in the feasible field size to be treated. In addition, many interesting features are local in nature and can be distributed spatially unequally, and are well represented only using a very large number of Laplacian eigenfunctions. For this reason, I introduce a local approach by defining small regions around each point in the domain and describe how to define, visualize, and analyze features in such a local neighborhood.

5.3.1 Locality and Local Feature Definition

I introduce a region $\epsilon(v_i)$ around each vertex v_i in the domain of interest. I then determine a local eigenbasis for this region and project only the local field inside the region on this local basis. From this, I obtain a local feature vector for every point in space consisting of the basis coefficients of the local field around each point, namely each vertex for a discretized field.

By using the same local region discretization for every point in space, only a single (small) eigenbasis must be computed and can be reused for every region. As long as the region's spatial resolution is high enough, scaling the region around each vertex can be done without recomputing the eigenvalue system.

Figure 5.3 illustrates these relationships. For a region of interest in a vector field, a local vector field basis is determined by the Laplacian eigenbasis fields corresponding to the region. Successively, the vector field is projected onto the elements of the eigenbasis, resulting in basis coefficients. These are then interpreted as a local feature descriptor or feature vector to define features and corresponding features strengths in the vector field as follows.

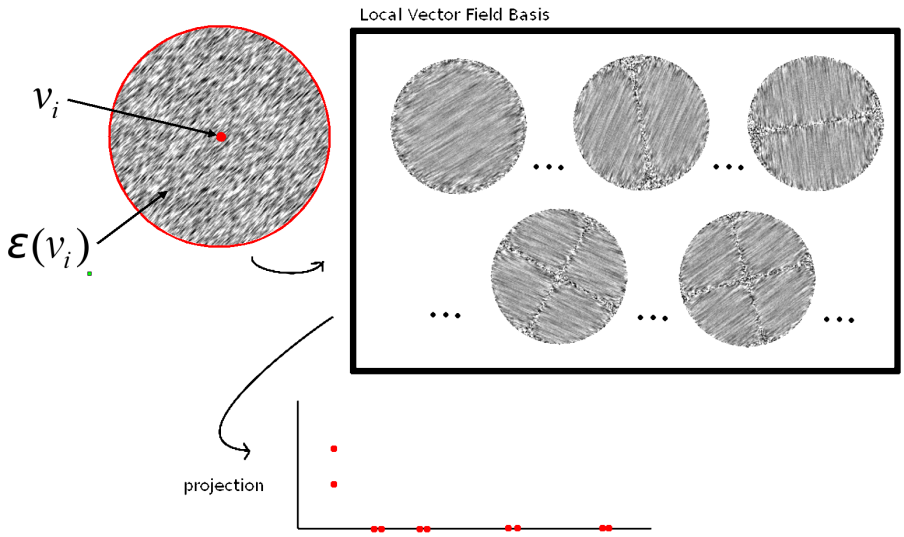


Figure 5.3: Local Feature Definition using local eigenvector basis. For the region $\epsilon(v_i)$ around v_i the Laplacian eigenfunctions are determined. By projecting the original field onto these eigenfunctions the vector of basis coefficients is composed.

A feature f is defined by a feature strength function ρ_f, which is a mapping from the Laplacian eigenbasis coefficients a_i to the interval $[0, 1]$:

$$\rho_f : \mathbb{R}^n \rightarrow \mathbb{R} \tag{5.1}$$

$$(a_1, a_2, \ldots, a_n) \mapsto [0, 1] \tag{5.2}$$

Here, resulting feature strength of 0 implies a vanishing response to the feature type f in the selected region while a resulting value of 1 implies the definition matches exactly.

Consider the following example. The first eigenfield in the local eigenbasis for the example vector field introduced above is shown in Figure 5.3. Together with an orthogonal eigenfield of the same eigenvalue, they form the first eigenspace. As depicted in the figure, the first eigenspace field contains exactly the linear component of the vector field. Therefore, I define the local vector field linearity ρ_L as a feature strength function by the ratio of the first two basis coefficients to the remainder of the basis coefficients on the spectrum as

$$\rho_L(a_1, \ldots, a_n) := \frac{\sum_{i=1}^{2} |a_i|}{\sum_{i=1}^{n} |a_i|}.$$

The purpose of this definition of ρ_L is to capture the relative importance of the basis functions that encode linear flow – as given by their coefficients a_1 and a_2 – normalized by the overall size of the coefficients ($\sum_{i=1}^{n} |a_i|$). As ρ_L approaches 1, the contribution of non-linear eigenbasis elements represented by $a_3, \ldots, a_n$ necessarily tends to zero, signifying diminishing non-linear behavior.

In this case depicted in Figure 5.3, only the first two basis coefficients of the local field basis are non-zero, and the other coefficients vanish, thus the field is perfectly linear in the considered neighborhood.

In other words, the feature strength functions measures the similarity of the feature vector to a desired feature type, where both are expressed in the Laplacian eigenbasis.

Naturally, local feature strength criteria can be applied globally for each point in the domain of the considered field. For example, Figure 5.4 (left) illustrates the local

linearity feature strength for the inflow dataset. High local linearity is colored in green while low local linearity is shown in red. The turbulent regions, where linear flow is not prevalent, are clearly distinguishable from the remainder of the field. In the right image, a simple threshold on ρ_L is used to perform a binary segmentation and separate mostly local linear regions from the turbulent parts of the vector field.

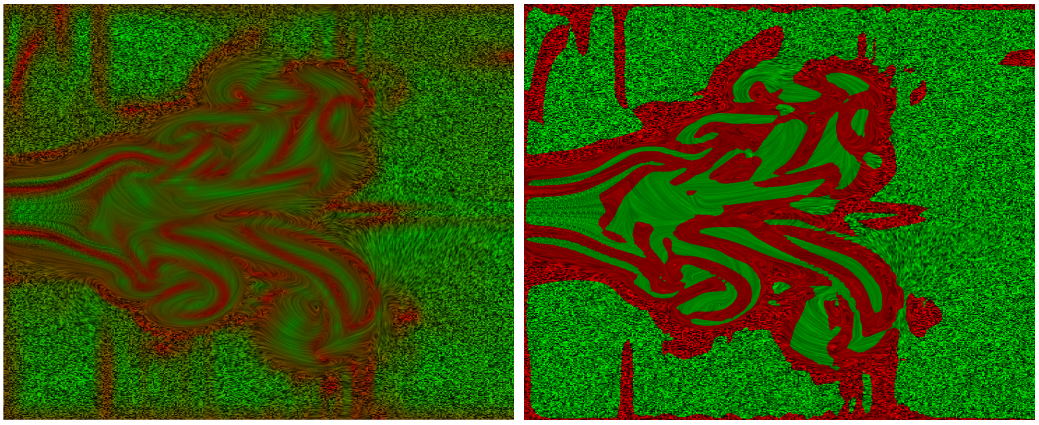


Figure 5.4: Application of local feature definitions with local vector field linearity on the left and its segmentation on the right.

5.4 Discretizations and Computational Issues

The harmonic analysis technique can be used on arbitrary field and domain types. However, the choice of Laplacian discretization is crucial to guarantee good results. In the following, I summarize the most important smooth Laplacian versions, followed by a brief introduction to a finite element and a discrete exterior calculus discretization and their comparison. Finally, I will discuss computational costs.

5.4.1 Finite Element discretization

To translate the concepts described above into practice, a discretization of the Laplacian operator acting over an unstructured mesh must be chosen. The simplest choice for this problem is finite element (FEM) formulation to discretize the Laplacian on the global domain or the local region. In this setting, the action of the Laplacian

on a vector field is approximated by its action on interpolatory basis functions that form a basis of the vector field function space over the discrete domain. For example, for linear finite elements over triangles covering a two-dimensional domain, the interpolation functions are simple hat functions defined over the one-ring neighborhood of a vertex. For vector fields, there is one basis function per vertex and per vector component.

For a given domain Ω and the Laplace eigenvalue problem

$$\begin{aligned} \Delta u - \lambda u &= 0 \qquad \text{in } \Omega \quad (5.3) \\ u &= 0 \qquad \text{on } \partial\Omega \quad (5.4) \end{aligned}$$

an approximation $\tilde{u}$ for the function u is sought according to Galerkin's method (cf. [Gal15, Fle84] by linearly independent functions $N_1, \ldots, N_m$ satisfying the homogeneous boundary condition

$$\tilde{u} = \sum_{k=1}^{m} u_k N_k. \quad (5.5)$$

In this formulation, the residual $||u - \tilde{u}||$, weighted with certain weight functions, is required to vanish. In Galerkin's method the basis functions $N_1, \ldots, N_m$ are reused as weight functions, and one obtains

$$\int_\Omega (\Delta\tilde{u} + \lambda\tilde{u}) N_j d\Omega = 0 \qquad \forall j = 1, \ldots, m. \quad (5.6)$$

Applying Green's formula and inserting (5.5) the Laplace eigenvalue problem is stated as the eigenvalue problem

$$\sum_{i=1}^{m} u_i \int_\Omega \nabla N_i \cdot \nabla N_j d\Omega = \lambda \sum_{i=1}^{m} u_i \int_\Omega N_i N_j d\Omega \qquad \forall j = 1, \ldots, m, \quad (5.7)$$

that is equivalent to the matrix eigenvalue problem

$$Au = \lambda M u \quad (5.8)$$

with the matrix coefficients

$$a_{ij} = \int_\Omega \nabla N_i \cdot \nabla N_j d\Omega \quad \text{and} \quad m_{ij} = \int_\Omega N_i N_j d\Omega.$$

5.4.2 Discrete Exterior Calculus (DEC) discretization

A general framework for exterior calculus on discrete manifolds is the Discrete Exterior Calculus (DEC) framework introduced in [Hir03]. It has been used for a variety of tasks in a variety of settings such as mesh processing [VL08b], fluid simulation [ETK+07], and others, and is conceptually straightfoward. Since it is less well known than the FEM approach, I provide a brief overview of the fundamental concepts in the following.

In the DEC framework, the geometry of discrete manifolds is described using simplicial complexes which are constructed from simplices. k-simplices have $k+1$ vertices, that is 0-simplices are vertices, 1-simplicles represent edges, 2-simplices are triangles, and 3-simplices are tetrahedra of a mesh. I make use of this framework to investigate a discrete version of the smooth Laplace-de Rham operator $\Delta = \mathrm{d}\delta + \delta\mathrm{d}$.

A central concept in exterior calculus is the concept of differentiable k-forms, differentiable forms that capture the notion of integrability over (sub-) manifolds of corresponding dimension (cf. [FH89, BG68]). For example, a 1-form – which can be used to describe a vector field – can be integrated over a 1-dimensional submanifold or curve, thus describing a line integral.

The DEC framework is intimately tied to this notion and directly represents the integrals of k-forms for each k-simplex of the discrete manifold, i.e. one scalar value is assigned to each vertex, edge, triangle, and tetrahedron of a discretized manifold in 3-space. Therefore, scalar functions can be mapped to scalar values on vertices, vector valued functions as 1-forms are mapped to scalar values on edges and so on.

The implementation of the exterior derivative d uses Stokes' theorem, giving a unique relationship between the exterior derivative d and the boundary ∂ of a k-form ω on a region σ with

$$\int_\sigma d\omega = \int_{\partial\sigma} \omega.$$

In other words, the evaluation of the exterior derivative d can be interpreted as the evaluation on the boundary of the same simplex.

Thus, the exterior derivative d on k-forms can be computed as a $K^{k+1} \times K^k$-matrix D^k, where K^i is the number of i-simplices in the simplicial complex. This matrix D^k is the transposed incidence matrix of $k+1$-simplices and k-simplices.

The implementation of the DEC framework requires a second entity from exterior calculus, the Hodge star $\star$. In a simplicial complex, each cell has a dual cell. For example, in a tetrahedral mesh contained in a 3-dimensional embedding, each tetrahedron has a dual vertex. The Hodge star maps a k-form to the complementary $(n-k)$-form on the corresponding dual cell. Since it is natural to require an integral to be proportional to the volume of its domain of integration, the Hodge star can be defined with the relation of dual and primal volumes,

$$\star = \frac{\text{vol(dual)}}{\text{vol(primal)}}.$$

In matrix representation, this gives a diagonal $K^k \times K^k$-matrix H^k with the fractions of dual and primal volumes as matrix entries.

The application of all possible exterior derivatives and Hodge stars to each k-form results in a discrete version of the de Rham complex:

$$\begin{array}{ccccccccccccc}
0 & \longrightarrow & \Omega^0 M & \xrightarrow{d} & \Omega^1 M & \xrightarrow{d} & \Omega^2 M & \xrightarrow{d} \ldots \xrightarrow{d} & \Omega^n M & \longrightarrow & 0 \\
 & & \downarrow \star & & \downarrow \star & & \downarrow \star & & \downarrow \star & & \\
0 & \longleftarrow & \Omega^n M & \xleftarrow{d} & \Omega^{n-1} M & \xleftarrow{d} & \Omega^{n-2} M & \xleftarrow{d} \ldots \xleftarrow{d} & \Omega^0 M & \longleftarrow & 0.
\end{array}$$

This finally allows the definition of the co-derivative δ for every $(k+1)$-form as the inverse mapping to the exterior derivative d with

$$\delta = (-1)^{n \cdot k+1} \star d \star .$$

Finally, the discrete Laplace-de Rham operator L, approximating the smooth Laplace-de Rham operator $\Delta = \mathrm{d}\delta + \delta\mathrm{d}$, can be implemented by a concatenation of matrices D and H on the primal and dual simplicial complex. Extended descriptions about DEC and its implementations can be found in [DKT06, ES05b].

5.4.3 Comparison of FEM and DEC discretizations

It is not a priori obvious which discretization – FEM or DEC – approach gives the best results for harmonic analysis of vector fields as described above. Furthermore, it can be shown that there is no discretized version of the Laplacian operator that captures all the properties of the corresponding continuous one [WMKG08]. Hence, I evaluate different discretization with respect to their suitability towards harmonic analysis in a discrete setting.

To obtain a qualitative understanding of the approximation qualities of FEM and DEC that can inform the choice of discretization method used in local feature analysis, I define a simple test problem. I consider a rectangular region in the plane $\Omega = [0, a] \times [0, b]$ with edge lengths $a, b > 0$ and the scalar eigenvalue problem

$$\begin{aligned} \Delta u - \lambda u &= 0 \qquad \text{in } \Omega \qquad (5.9) \\ u &= 0 \qquad \text{on } \partial\Omega \qquad (5.10) \end{aligned}$$

as a test case for which an analytic solution is known. This problem can be understood as a tensor product of the Fourier decomposition described in Section 5.2.1. Here, the eigenfunctions e_{ij} are products of sinusoidal functions and the eigenvalues λ_{ij} are sums of the corresponding one-dimensional eigenvalues, namely

$$\begin{aligned} e_{ij}(x, y) &= \sin\left(\frac{i\pi x}{a}\right) \sin\left(\frac{j\pi y}{b}\right), \\ \lambda_{i,j} &= \pi^2 \left[\left(\frac{i}{a}\right)^2 + \left(\frac{j}{b}\right)^2\right], \\ i, j &= 1, 2, \ldots. \end{aligned}$$

In the following, I compare the numerically approximated eigenvalues and eigenfunctions obtained by the FEM and DEC approaches to these analytical solutions.

Table 5.1 gives the eigenvalues of Eq. 5.9, as numerically determined from the FEM and DEC discretizations of a square region ($a, b = \pi$), with increasing grid resolution. It is observable, that the convergence to the correct eigenvalue known from the analytic solution is better for the finite element method. Furthermore, even for

relatively higher grid resolution the DEC approach seems to diverge to a slightly different value. This is an important observation for the situation in which eigenvalues are used for harmonic analysis (cf. Section 5.3.1).

Figure 5.5 illustrates the eigenfunctions of Eq. 5.9. The first eigenfunction as well as one of the two fourth eigenfunctions are shown using a color map on the left. The middle and right columns illustrate the pointwise error of the FEM and DEC approximations, respectively, using a greyscale mapping.

The error distribution over the squares is very similar for both approximation methods, and they appear qualitatively equivalent for the given test case. In general, not much difference between the FEM and DEC solution is observed. However, for the first eigenfunction, the FEM solution is consistently better due to the property that FEM minimizes the integral error over the domain.

In summary, eigenfunctions computed by each of the two discretizations are quite similar. Therefore, the FEM approach as well as the DEC approach can be used in cases focusing on the usage of eigenfunctions. As an example, this is true for the local vector field linearity feature strength function rho_L I define in Section 5.3.1.

However, my comparison shows a clear advantage of the FEM discretization in approximating eigenvalues. For this reason, in cases relying on accurate eigenvalues the FEM approach should be preferred.

In a general setting, there are also other factors to consider when choosing a discretization for the Laplacian. While applying different boundary conditions is often possible in a finite element approach, the adaptation to arbitrary embeddings is not

analytic	FEM				DEC			
	11x11	25x25	51x51	101x101	11x11	25x25	51x51	101x101
2	2,0441	2,0076	2,0017	2,0004	2,1234	2,1514	2,1560	2,1570
5	5,2408	5,0412	5,0095	5,0023	5,1808	5,3559	5,3847	5,3912
8	8,7147	8,1223	8,0281	8,0070	8,1002	8,5349	8,6076	8,6240
10	10,8610	10,1456	10,0334	10,0083	9,9416	10,6356	10,7518	10,7781

Table 5.1: The eigenvalues of the finite element and the discrete exterior calculus discretizations on increasing grid resolutions are compared to the ones known from calculus. The finite element method is found to deliver more accurate results.

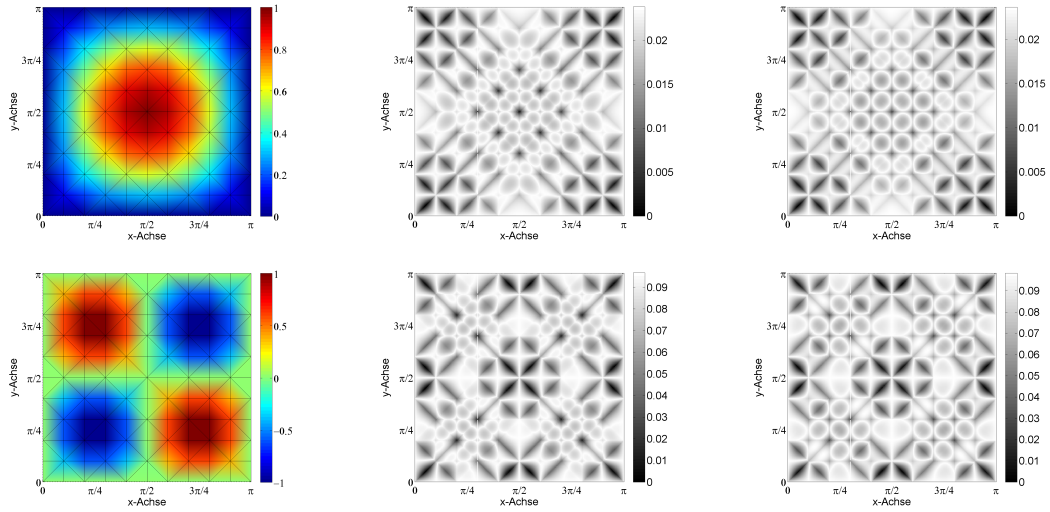


Figure 5.5: The point-wise error of the finite element field solutions (middle) and the exterior calculus field solutions (right) compared to the analytic solution (left) on a 11×11-grid.

trivial. Conversely, this embedding is trivial in the discrete exterior calculus setup, since its formulation only depends on simplex incidences and automatically handles cases like a two-dimensional surface in an arbitrary three-dimensional embedding.

5.4.4 Computation of Large Eigenvalue Sets

Solving a matrix eigenvalue system for its eigenvalues and eigenvectors is typically a time-consuming task. In addition, the amount of memory required to represent the corresponding matrices grows quadratic with the number of grid points or variables. This makes harmonic field analysis computationally expensive and must be considered a bottleneck – especially for global approaches where many eigenvectors must be computed.

[VL08a] combines the shift-and-invert method with band-by-band computation for manifold harmonics. This method can also be used for harmonic field analysis since it is a general method applicable to any matrix.

In a nutshell, the shift-and-invert method is a common spectral transformation. Instead of computing the eigenvalues and eigenvectors around an eigenvalue σ, the

spectrum can be shifted such that only computing the eigenvalues around zero is needed by the examination of $A - \sigma I x$ instead of the initial matrix A. Since most algorithms for eigenvalue computation perform best for large eigenvalues modulus, the spectrum can additionally be inverted by inverting the matrix. Finally, not the eigenvalue λ is computed but the modified eigenvalue Θ with

$$A_\sigma^{-1} x = (A - \sigma I)^{-1} x = \Theta x, \qquad \Theta = \frac{1}{\lambda - \sigma}.$$

To solve this system, the shift-and-invert method, e.g. as implemented in ARPACK, is used. This interface is accelerated using SUPERLU for matrix vector operations and LU matrix decomposition. There is no actual need to compute and store the inverted matrix, because the computation only requires the evaluation of $x = A_\sigma^{-1} z$ for an arbitrary vector z. This can be computed very efficiently with the LU decomposition of A_σ by performing a back-substitution.

The entire spectrum corresponding to the Laplacian discretization matrix can then be computed in multiple parts. A small band of eigenvalues is computed at once; successively, this band is shifted with a small overlap until the entire spectrum is completed. With this technique, the sub-linear behavior of eigenvalue solvers can also be compensated.

In my experiments, the global eigenvalue problem for the inflow dataset with 36000 edge values was solved in about 2 hours on a commodity PC (Intel Q6600 quad-core machine at 2.4 GHz). Also applying classical eigenvalue solvers, I was only able to determine the first 3000 eigenvectors in about 3 days of computation time. In comparison, for the two-dimensional local field analyses examples in Section 5.3.1 all the six to ten vector field basis functions were determined in less than 100 ms on the same hardware.

5.5 Conclusion

Harmonic analysis techniques are a fundamental tool in scalar field processing and analysis. Well-known applications are techniques based on the Fourier decomposition. In this chapter, I provide a number of ideas on applying harmonic analysis on

scalar, vector and tensor fields over general domains, with a focus on visualization and analysis applications.

The typical global approach to harmonic analysis was illustrated on a small vector field example given an unstructured grid, and the problems for global approaches were identified and discussed. These issues led us to the introduction of a local approach using local regions for every point in space. I used this local approach to define a local feature strength measure, based on properties described in Laplacian eigenbases.

Since two direct possibilities for the choice of the discretization are apparent, I gave an overview of both methods and compared their behavior numerically. In addition, I discussed numerical aspects of the eigenvalue computation that has proven difficult to master in my experiments.

For the future, there are still many open avenues of investigation. Besides improving the computational system in accuracy, convergence and computational costs, for special cases there might be explicit algorithms not depending on a numerical eigensolver. Furthermore, other possibilities for feature measures going beyond the local vector field linearity criterion should be explored, and it is conceivable to apply the described technique to achieve pattern matching of empirically defined features. Finally, the adaptation of existing signal and image processing algorithms should also be considered.

Chapter 6

Conclusion

The overall topic of this thesis was the preparation of massive parallel CFD simulations for computational steering and the development of new interactive online monitoring techniques. Since the computational steering loop turned out to be a complex task involving system design, user interaction and algorithmic components contributions have been made in all of these areas.

I introduced FSSteering, a flexible computational steering framework enabling to easily steer existing and new simulations based on the FlowSimulator framework. I was able to show its usage throughout this thesis and showed in two steering examples the capability to steer running simulations. Being highly adaptive, sophisticated online monitoring examples could be presented throughout my thesis.

In order to enable for interactive exploration of running simulations, I presented an interactive cut-plane online monitoring approach. By implementing this approach into the FSSteering framework, the approach could extract information to visualize in-situ without the need to duplicate any data, which can be expensive for large-scale simulations. Since update rates and latency is crucial for interactive exploration methods, I demonstrated the flaws of traditional cell-based cut-plane extraction algorithms. By using point-based sampling, I could provide a method which achieved these requirements and enabled to explore scalar fields of ongoing simulations in a coupled virtual environment.

With more complex visualization features and ever increasing simulation scales, local visualization front-ends can easily be overwhelmed by their sizes and can not provide rendering at interactive frame rates anymore. In this work, I presented a hybrid rendering approach combining local and parallel remote rendering hardware which removes the heavy duty rendering workload from local resources such as virtual environment hardware. Therefore, my approach uses a local context geometry for navigation purposes which can be easily rendered locally. The heavy rendering workload is carried out on distant remote rendering hardware. This approach decouples both rendering processes and results in different update rates as well as latency for local and remote images. With the presented re-adjustment step I could clearly show that this approach leads to successful navigation through large-scale data sets. Two challenges for this approach are limited bandwidth and providing sufficient update rates for time-dependent features. For these challenges solutions were presented. Progressive image streaming is able to provide adaptive image resolutions for high pixel-count display such as tiled display walls. Furthermore, I demonstrated that utilizing the parallelism features of modern CPUs with pipelining and extending my approach to use multiple render instances, I was able to provide high update rates sufficient for time-dependent simulation results with high temporal resolution.

Finally, feature extraction itself becomes more important as well as more complex for large-scale simulation results. Existing methods are typically based on deep mathematical frameworks or require intimate knowledge of the application domain. Here, I presented the application of harmonic analysis to vector fields as an adaptable, mathematical framework for feature-based visualization. The definition of a feature space over small neighborhoods enabled to define, locate and compare local features in an empirical fashion.

Bibliography

[ADD+07] Matt Aranha, Piotr Dubla, Kurt Debattista, Thomas Bashford-Rogers, and Alan Chalmers. A physically-based client-server rendering solution for mobile devices. In *Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*, MUM '07, pages 149–154, New York, NY, USA, 2007. ACM.

[AK08] Ingo Assenmacher and Torsten Kuhlen. The ViSTA virtual reality toolkit. In *Proceedings of the SEARIS Workshop*, IEEE VR 2008. Shaker Verlag, 2008.

[ARS11] James Ahrens, David Rogers, and Becky Springmeyer. Visualization and data analysis at the exascale. National Nuclear Security Administration (NNSA) Accelerated Strategic Computing (ASC) Exascale Environment Planning Process, 2011.

[BCE11] Javier Bartolome Calvo and Thino Eggers. Application of a coupling of aerodynamics and flight dynamics to the SHEFEX I flight experiment. In *In AIAA-2011-2323*, April 2011.

[Bea96] David M. Beazley. Swig: an easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the 4th conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, pages 15–15, Berkeley, CA, USA, 1996. USENIX Association.

[Ber99] M. Berzins. Mesh quality: A function of geometry, error estimates or both? *Engineering with Computers*, 15:236–247, 1999. 10.1007/s003660050019.

[BG68] R.L. Bishop and S.I. Goldberg. *Tensor analysis on manifolds*. Dover Publications, 1968.

[BL10] Tarik Barth and Jose Longo. Advanced flight analysis of SHEFEX-I. In *New Results in Numerical and Experimental Fluid Mechanics VII*, volume 112 of *Notes on Numerical Fluid Mechanics and Multidisciplinary*

Design, pages 431–439. Springer Verlag, Berlin Heidelberg, October 2010.

[Bou97] Paul Bourke. Polygonising a scalar field, June 1997.

[Bry96] Steve Bryson. in scientific visualization. *Virtual Reality*, 39(5):62–71, 1996.

[BSF10] S. Beck, M. Schneider, and B. Fröhlich. Multiple view generation for auto-stereoscopic displays. In *Proceedings of the 7. Workshop on "Virtuelle und Erweiterte Realität der GI-Fachgruppe VR/AR"*, pages 21–32. Shaker, 2010.

[BSO+12] John Biddiscombe, Jerome Soumagne, Guillaume Oger, David Guibert, and Jean-Guillaume Piccinali. Parallel computational steering for hpc applications using hdf5 files in distributed shared memory. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):852–864, 2012.

[CDE03] O. Coulaud, M. Dussere, and A. Esnard. Toward a distributed computational steering environment based on corba, 2003.

[CW93] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 279–288, New York, NY, USA, 1993. ACM.

[CWF+12] Fang Chen, Christian Wagner, Markus Flatken, Andreas Gerndt, and Hans Hagen. Enabling interactive mesh quality exploration of large scale CFD simulations in virtual environments. Poster, 2nd Symposium on Large Scale Data Analysis and Visualization (LDAV), October 2012.

[DCH88] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. *SIGGRAPH Comput. Graph.*, 22(4):65–74, June 1988.

[DGL97] James W. Demmel, John Gilbert, and Xiaoye S. Li. SuperLU users' guide. Technical Report CSD-97-944, 8, 1997.

[DKT06] Mathieu Desbrun, Eva Kanso, and Yiying Tong. Discrete differential forms for computational modeling. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, pages 39–54, New York, NY, USA, 2006.

[DP11] S. Dragomir and D. Perrone. *Harmonic Vector Fields: Variational Principles and Differential Geometry*. Elsevier Science Ltd, 2011.

[DtBG06] Shen Dong, Peer timo Bremer, and Michael Garl. Spectral surface quadrangulation. *ACM Transaction on Graphics*, 25:1057–1066, 2006.

[Dur99] Lisa Durbeck. Evaporation: a technique for visualizing mesh quality. In *Proceedings of the 8th International Meshing Roundtable, South Lake Tahoe*, pages 11–13, 1999.

[EDC04] Aurélien Esnard, Michaël Dussere, and Olivier Coulaud. A time-coherent model for the steering of parallel simulations. In *Europar 2004*, pages 90–97. Springer Verlag, 2004.

[EEH+00] K. Engel, T. Ertl, P. Hastreiter, B. Tomandl, and K. Eberhardt. Combining local and remote visualization techniques for interactive volume rendering in medical applications. In *Proceedings of the conference on Visualization '00*, VIS '00, pages 449–452, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[EFG+05] Th. Eickermann W. Frings, P. Gibbon, L. Kirtchakova, D. Mallmann, and A. Visser. Steering unicore applications with visit. *Philosophical Transactions of The Royal Society. Journal*, 2005.

[EHT+00] K. Engel, P. Hastreiter, B. Tomandl, K. Eberhardt, and T. Ertl. Combining local and remote visualization techniques for interactive volume rendering in medical applications. In *Visualization 2000. Proceedings*, pages 449 –452, oct. 2000.

[EMP09] S. Eilemann, M. Makhinya, and R. Pajarola. Equalizer: A scalable parallel rendering framework. *Visualization and Computer Graphics, IEEE Transactions on*, 15(3):436 –452, may-june 2009.

[ES03] J. Ebling and G. Scheuermann. Clifford convolution and pattern matching on vector fields. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 26. IEEE Computer Society, 2003.

[ES05a] J. Ebling and G. Scheuermann. Clifford Fourier transform on vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):469–479, 2005.

[ES05b] Sharif Elcott and Peter Schröder. Building your own dec at home. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 8, New York, NY, USA, 2005.

[ETK+07] Sharif Elcott, Yiying Tong, Eva Kanso, Peter Schröder, and Mathieu Desbrun. Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph.*, 26, January 2007.

[FH89] Flanders and Harley. *Differential forms with applications to the physical sciences*. Dover Publications, 1989.

[Fle84] C.A.J. Fletcher. *Computational Galerkin methods*. Springer series in computational physics. Springer-Verlag, 1984.

[FMA05] Nathaniel Fout, Kwan-Liu Ma, and James Ahrens. Time-varying, multivariate volume data reduction. In *Proceedings of the 2005 ACM symposium on Applied computing*, SAC '05, pages 1224–1230, New York, NY, USA, 2005. ACM.

[FMT+11] N. Fabian, K. Moreland, D. Thompson, A.C. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K.E. Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 89–96. IEEE, 2011.

[FS07] Matthew Fisher and Peter Schröder. Design of tangent vector fields. *ACM Trans. Graph*, 26:56, 2007.

[Gal15] Boris Grigoryevich Galerkin. On electrical circuits for the approximate solution of the laplace equation. *Vestnik Inzh.*, 19:897–908, 1915.

[GHW+04] Andreas Gerndt, Bernd Hentschel, Marc Wolter, Torsten Kuhlen, and Christian Bischof. Viracocha: An efficient parallelization framework for large-scale cfd post-processing in virtual environments. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 50, Washington, DC, USA, 2004. IEEE Computer Society.

[GJ10] Christoph Garth and Ken Joy. Fast, memory-efficient cell location in unstructured grids for visualization. *IEEE Transactions on Computer Graphics and Visualization*, 16(6):1541–1550, November 2010.

[GSJ+06] Jinghua Ge, Daniel J. Sandin, Andrew Johnson, Tom Peterka, Robert Kooima, Javier I. Girado, and Thomas A. DeFanti. Point-based vr visualization for large-scale mesh datasets by real-time remote computation. In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, VRCIA '06, pages 43–50, New York, NY, USA, 2006. ACM.

[GSRB01] Desmond Germans, Hans J. W. Spoelder, Luc Renambot, and Henri E. Bal. Virpi: A high-level toolkit for interactive scientific visualization in virtual reality. In *Proc. Immersive Projection Technology/Eurographics Virtual Environments Workshop*, 2001.

[GW06] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[HDSB08] K.H. Huebner, D.L. Dewhirst, D.E. Smith, and T.G. Byrom. *The finite element method for engineers*. Wiley India Pvt. Ltd., 2008.

[Hea02] M.T. Heath. *Scientific computing*. McGraw-Hill, 2002.

[HEB+01] Greg Humphreys, Matthew Eldridge, Ian Buck, Gordon Stoll, Matthew Everett, and Pat Hanrahan. Wiregl: A scalable graphics system for clusters. In *Computer Graphics (Proceedings of SIGGRAPH 01*, 2001.

[HHN+02] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: A stream-processing framework for interactive rendering on clusters, 2002.

[Hir03] A.N. Hirani. *Discrete exterior calculus*. PhD thesis, 2003.

[HM99] Thomas C. Hudson and William R. Mark. Multiple image warping for remote display of rendered images. Technical report, Chapel Hill, NC, USA, 1999.

[Jay73] N. S. Jayant. Adaptive quantization with a one-word memory. *Bell System Technical Journal*, 52:‘1119–1144, 1973.

[JB10] Domenic Jenz and Martin Bernreuther. The computational steering framework steereo. In *Para 2010*, 2010.

[JC06] Gunnar Johansson and Hamish Carr. Accelerating marching cubes with graphics hardware. In *In CASCON '06: Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research, ACM*, page 378. Press, 2006.

[Kat04] Y. Katznelson. *An introduction to harmonic analysis*. Cambridge mathematical library. Cambridge University Press, 2004.

[KBLH03] Oliver Kreylos, E Wes Bethel, Terry J Ligocki, and Bernd Hamann. Virtual-reality based interactive exploration of multiresolution data. *Hierarchical and Geometrical Methods in Scientific Visualization*, pages 1–20, 2003.

[Knu01] Patrick M. Knupp. Algebraic mesh quality metrics. *SIAM J. Sci. Comput.*, 23(1):193–218, January 2001.

[Kre78] E. Kreyszig. *Introductory functional analysis with applications*. Wiley New York, 1978.

[KSE04] Thomas Klein, Simon Stegmaier, and Thomas Ertl. Hardware-accelerated reconstruction of polygonal isosurface representations on unstructured grids. In *In Proceedings of Pacific Graphics '04*, pages 186–195, 2004.

[KTH+02] O. Kreylos, A. M. Tesdall, B. Hamann, J. K. Hunter, and K. I. Joy. Interactive visualization and steering of cfd simulations. In S. Müller and W. Stärzlinger, editors, *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*, pages 25–34, 2002.

[LBG80] Y Linde, A Buzo, and R Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, 1980.

[LC87] W E Lorensen and H E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer*, 21(4):163–169, 1987.

[LHA01] C. Charles Law, Amy Henderson, and James Ahrens. An application architecture for large data visualization: a case study. In *Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, PVG '01, pages 125–128, Piscataway, NJ, USA, 2001. IEEE Press.

[LSY98] RB Lehoucq, DC Sorensen, and C. Yang. ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods. Technical report, SIAM, Philadelphia, 1998.

[Ma07] Kwan-Liu Ma. Emerging visualization technologies for ultra-scale simulations. *CTWatch Quarterly*, 3(4):26 –52, 2007.

[MB95] Leonard McMillan and Gary Bishop. Head-tracked stereoscopic display using image warping. In *PROCEEDINGS SPIE, VOLUME 2409*, pages 21–30, 1995.

[MCEF08] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. In *ACM SIGGRAPH ASIA 2008 courses*, SIGGRAPH Asia '08, pages 35:1–35:11, New York, NY, USA, 2008. ACM.

[ME10] Michael Meinel and Gunnar Ólafur Einarsson. The flowsimulator framework to unify massively parallel cfd applications. In *Para 2010*, 2010.

[MM07] C. Muelder and K.-L. Ma. Rapid feature extraction and tracking through region morphing. Technical report, Computer Science Department, University of California at Davis, 2007.

[MMB97] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3d warping. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, I3D '97, pages 7–ff., New York, NY, USA, 1997. ACM.

[MRH+07] Kwan-Liu Ma, Robert Ross, Jian Huang, Greg Humphreys, Kenneth Morel, John D. Owens, and Han wei Shen. Ultra-scale visualization: Research and education, 2007.

[MvWL98] Jurriaan D. Mulder, Jarke van Wijk, and Robert Van Liere. A survey of computational steering environments. *Future Generation Computer Systems*, 13, 1998.

[MWP01] Kenneth Moreland, Brian Wylie, and Constantine Pavlakos. Sort-last parallel rendering for viewing extremely large data sets on tile displays. In *Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, PVG '01, pages 85–92, Piscataway, NJ, USA, 2001. IEEE Press.

[MWYT07] Kwan-Liu Ma, Chaoli Wang, Hongfeng Yu, and Anna Tikhonova. In-situ processing and visualization for ultrascale simulations. *Journal of Physics: Conference Series*, 78(1):012043, 2007.

[NHM11] Braden Neal, Paul Hunkin, and Antony McGregor. Distributed opengl rendering in network bandwidth constrained environments. In Torsten Kuhlen, Renato Pajarola, and Kun Zhou, editors, *EGPGV*, pages 21–29. Eurographics Association, 2011.

[NSOJA10] José M. Noguera, Rafael J. Segura, Carlos J. Ogayar, and Robert Joan-Arinyo. Navigating large terrains using commodity mobile devices. *Computers and Geosciences*, In Press, Corrected Proof:–, 2010.

[NSOJA11] José M. Noguera, Rafael J. Segura, Carlos J. Ogayar, and Robert Joan-Arinyo. Navigating large terrains using commodity mobile devices. *Computers & Geosciences*, 37(9):1218 – 1233, 2011.

[Pas04] V. Pascucci. Isosurface computation made simple: Hardware acceleration, adaptive refinement and tetrahedral stripping. In *In Joint Eurographics - IEEE TVCG Symposium on Visualization (VisSym*, pages 293–300, 2004.

[PNP05] Sébastien Piccand, Rita Noumeir, and Eric Paquette. Efficient visualization of volume data sets with region of interest and wavelets. In *SPIE Medical Imaging*, 2005.

[Pro08] The VirtualGL Project. A study of the performance of virtualgl 2.1 and turbovnc 0.4. Technical report, May 2008.

[PWJ97] Steven G. Parker, David M. Weinstein, and Christopher R. Johnson. The scirun computational steering software system, 1997.

[RCMS] M. Rivi, L. Calori, G. Muscianisi, and V. Slavnic. In-situ visualization: State-of-the-art and some use cases.

[RT06] G. Rong and T.S. Tan. Jump flooding in gpu with applications to voronoi diagram and distance transform. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 109–116. ACM, 2006.

[RW06] Jonathan C. Roberts and Michael A. E. Wright. Towards ubiquitous brushing for information visualization. In *Proceedings of the conference on Information Visualization*, IV '06, pages 151–156, Washington, DC, USA, 2006. IEEE Computer Society.

[SG96] Dieter Schmalstieg and Michael Gervautz. Demand-driven geometry transmission for distributed virtual environments. In *Computer Graphics Forum*, pages 421–433, 1996.

[SGvR+03] M. Schirski, A. Gerndt, T. van Reimersdahl, T. Kuhlen, P. Adomeit, O. Lang, S. Pischinger, and C. Bischof. Vista flowlib - framework for interactive visualization and exploration of unsteady flows in virtual environments. In *EGVE '03: Proceedings of the workshop on Virtual environments 2003*, pages 77–85, New York, NY, USA, 2003. ACM.

[SHA+09] Wolfram Schoor, Marc Hofmann, Simon Adler, Werner Benger, Bernhard Preim, , and Ruediger Mecke. Remote rendering strategies for large biological datasets. In *5th High-End Visualization Workshop, Baton Rouge, Louisiana*, 2009.

[SHM+07] M. Schlemmer, M. Heringer, F. Morr, I. Hotz, M. Hering-Bertram, C. Garth, W. Kollmann, B. Hamann, and H. Hagen. Moment Invariants for the Analysis of 2D Flow Fields. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1743, 2007.

[SLRF08] J.P. Springer, C. Lux, D. Reiners, and B. Froehlich. Advanced multi-frame rate rendering techniques. In *Virtual Reality Conference, 2008. VR '08. IEEE*, pages 177 –184, march 2008.

[SM99] H. Schumann and W. Müller. *Visualisierung: Grundlagen und allgemeine Methoden*. Springer, 1999.

[Sor96] D.C. Sorensen. Implicitly restarted Arnoldi/Lanczos methods for large scale eigenvalue calculations. *Institute for Computer Applications in Science and Engineering*, 1996.

[TACSD06] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. Designing quadrangulations with discrete harmonic forms. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, SGP '06, pages 201–210, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[THJ+06] John Turner, Marcus Hörschgen, Wolfgang Jung, Andreas Stamminger, and Peter Turner. SHEFEX - hypersonic re-entry fllight experiment - vehicle and subsystem design, flight performance and prospects. In *In Proceedings AIAA 14th Spaceplane Systems and Technologies Conference*, November 2006.

[Tom06] C. Tominski. *Event based visualization for user centered visual analysis*. 2006.

[TYRg+06] Tiankai Tu, Hongfeng Yu, Leonardo Ramirez-guzman, Jacobo Bielak, Omar Ghattas, Kwan liu Ma, and David R. O'Hallaron. From mesh generation to scientific visualization: an end-to-end approach to parallel supercomputing. In *Proceedings of ACM/IEEE Supercomputing 2006 Conference*, 2006.

[vDFL+00] A. van Dam, A.S. Forsberg, D.H. Laidlaw, Jr. LaViola, J.J., and R.M. Simpson. Immersive vr for scientific visualization: a progress report. *Computer Graphics and Applications, IEEE*, 20(6):26 –52, nov/dec 2000.

[vDLS02] Andries van Dam, David H Laidlaw, and Rosemary Michelle Simpson. Experiments in immersive virtual reality for scientific visualization. *Computers & Graphics*, 26(4):535 – 555, 2002.

[VIS05] Visit user's manual. Technical Report. UCRL-SM-220449, Lawrence Livermore National Laboratory, October 2005, October 2005.

[VL08a] Bruno Vallet and Bruno Levy. Spectral geometry processing with manifold harmonics. *Computer Graphics Forum (Proceedings Eurographics)*, 2008.

[VL08b] Bruno Vallet and Bruno Lévy. Spectral geometry processing with manifold harmonics. *Computer Graphics Forum (Proceedings Eurographics)*, 2008.

[WBK07] M. Wolter, Ch. Bischof, and T. Kuhlen. Dynamic regions of interest for interactive flow exploration. In *Proceedings of Parallel Graphics and Visualization 2007*, pages pp. 61–68, 2007.

[WFC+12] Christian Wagner, Markus Flatken, Fang Chen, Andreas Gerndt, Charles D. Hansen, and Hans Hagen. Interactive hybrid remote rendering for multi-pipe powerwall systems. In Christian Geiger, Jens Herder, and Tom Vierjahn, editors, *VR/AR*, pages 155–166. Shaker, 2012.

[WFM+10] Christian Wagner, Markus Flatken, Michael Meinel, Andreas Gerndt, and Hans Hagen. FSSteering: A distributed framework for computational steering in a script-based cfd simulation environment. Berlin, Lehmanns Media-LOB.de, 2010.

[WGH12] Christian Wagner, Christoph Garth, and Hans Hagen. Harmonic field analysis. In David H. Laidlaw and Anna Vilanova, editors, *New Developments in the Visualization and Processing of Tensor Fields*, Mathematics and Visualization, pages 363–379. Springer Berlin Heidelberg, 2012.

[WGHH12] Christian Wagner, Andreas Gerndt, Charles Hansen, and Hans Hagen. Interactive in-situ online monitoring of large scale cfd simulations with cut-planes. In *IEEE Virtual Reality Workshop, Immersive Visualization Revisited: Challenges and Opportunities*, March 2012.

[WHS+06] M. Wolter, B. Hentschel, M. Schirski, A. Gerndt, and T. Kuhlen. Time step prioritising in parallel feature extraction on unsteady simulation data. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization 2006*, 2006.

[WM08] Chaoli Wang and Kwan-Liu Ma. A statistical approach to volume data quality assessment. *Visualization and Computer Graphics, IEEE Transactions on*, 14(3):590–602, 2008.

[WMKG08] Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. Discrete laplace operators: no free lunch. In *SIGGRAPH Asia '08:*

ACM SIGGRAPH ASIA 2008 courses, pages 1–5, New York, NY, USA, 2008.

[WSK+07] Marc Wolter, Marc Schirski, Torsten Kuhlen, C. Bischof, M. Bücker, P. Gibbon, G. R. Joubert, B. Mohr, F. Peters (eds, Marc Wolter, Marc Schirski, and Torsten Kuhlen. Hybrid parallelization for interactive exploration in virtual environments, 2007.

[WWS+06] Z Wang, G Wu, H R Sheikh, E P Simoncelli, E Yang, and A C Bovik. Quality-aware images. *IEEE Trans Image Processing*, 15(6):1680–1689, Jun 2006.

BIOGRAPHICAL INFORMATION

Personal Details

Name	Christian Wagner
Date of birth	August 9, 1980
Place of birth	Trier, Germany
Nationality	German

Education

2008 – present	PhD student at the Department of Computer Science, Computer Graphics and HCI Group, University of Kaiserslautern, Germany
2008 – present	Stipend of the International Research Training Group (IRTG) "Visualization of Large and Unstructured Data Sets – Applications in Geospatial Planning, Modelling, and Engineering"
2001 – 2007	Study of Computer Science at the University of Kaiserslautern, Germany, Degree: Diplom-Informatiker (Dipl.-Inf.), grade: 1.2. Master Thesis: "Eigenanalysis of Vector Fields on Manifolds"
1991 – 2000	Max-Planck-Gymnasium Trier
1987 – 1991	Grundschule Kenn (Elementary School)

Work Experiences

10/2007–12/2007	"Entwicklungsagentur Rheinland-Pfalz e.V." - Maintenance of a web content management system
08/2004–06/2007	"Fraunhofer Institut für Techno- und Wirtschaftmathematik (ITWM)" - Work on a system for virtual fiber structure design including filtration simulation and visualization
04/2004–07/2004	Teaching assistant at the University Kaiserslautern - Supervision in the lecture "Grundlagen betrieblicher Informationssysteme" (Enterprise information systems)
10/2003–02/2004	Teaching assistant at the University Kaiserslautern - Supervision in the lecture "Entwicklung von Softwaresystemen III" (software design III)
10/2002–02/2003	Teaching assistant at the University Kaiserslautern - Supervision in the lecture "Rechnersysteme" (computer systems)

List of Publications

[**WFC+12**] Christian Wagner, Markus Flatken, Fang Chen, Andreas Gerndt, Charles D. Hansen, and Hans Hagen. **Interactive hybrid remote rendering for multi-pipe powerwall systems**. In Christian Geiger, Jens Herder, and Tom Vierjahn, editors, *VR/AR*, pages 155–166. Shaker, 2012.

[**WGHH12**] Christian Wagner, Andreas Gerndt, Charles Hansen, Hans Hagen. **Interactive In-Situ Online Monitoring of Large Scale CFD Simulations with Cut-Planes**. Short Paper, *IEEE Virtual Reality Workshop, Immersive Visualization Revisited: Challenges and Opportunities*, Orange County, CA, March 4, 2012.

[**WGH12**] Christian Wagner, Christoph Garth, and Hans Hagen. **Harmonic Field analysis**. In David H. Laidlaw and Anna Vilanova, editors, *New Developments in the Visualization and Processing of Tensor Fields, Mathematics and Visualization*, pages 363–379. Springer Berlin Heidelberg, 2012.

[**CWF+12**] Fang Chen, Christian Wagner, Markus Flatken, Andreas Gerndt, Hans Hagen. **Enabling Interactive Mesh Quality Exploration of Large Scale CFD Simulations in Virtual Environments**. Poster, *2nd Symposium on Large Scale Data Analysis and Visualization (LDAV)*, Seattle, Washington, USA, October 14–19, 2012.

[**WFM+10**] Christian Wagner, Markus Flatken, Michael Meinel, Andreas Gerndt, and Hans Hagen. **FSSteering: A distributed framework for computational steering in a script-based CFD simulation environment**. Berlin, Lehmanns Media-LOB.de, 2010.

[**Wag10**] ChristianWagner. **Open problems in computational steering of massive parallel unstructured grid based CFD simulations**. In *VLUDS'10*, pages 82–89, 2010.

[**MGH+09**] Robert Moorhead, Yanlin Guan, Hans Hagen, Sven Böttger, Natallia Kotava, and Christian Wagner. **Illustrative visualization: interrogating triangulated surfaces**. *Computing*, 86:131–150, 2009.

[**MSW+08**] Holger A. Meier, Michael Schlemmer, Christian Wagner, Andreas Kerren, Hans Hagen, Ellen Kuhl, and Paul Steinmann. **Visualization of particle interactions in granular media**. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):1110–1125, 2008.